EdgeTA
Neuron-Grained Scaling of Foundation Models in Edge-Side Retraining

Zhang, Qinglong; Han, Rui; Liu, Chi Harold; Wang, Guoren; Guo, Song; Chen, Lydia Y.

DOI
10.1109/TMC.2024.3504859

Publication date
2025

Document Version
Final published version

Published in
IEEE Transactions on Mobile Computing

EdgeTA: Neuron-Grained Scaling of Foundation Models in Edge-Side Retraining

Qinglong Zhang, Rui Han, Chi Harold Liu, *Senior Member, IEEE*, Guoren Wang, *Senior Member, IEEE*, Song Guo, *Fellow, IEEE*, and Lydia Y. Chen, *Senior Member, IEEE*

***Abstract*—Foundation models (FMs) such as large language models are becoming the backbone technology for artificial intelligence systems. It is particularly challenging to deploy multiple FMs on edge devices, which not only have limited computational resources, but also encounter unseen input data from evolving domains or learning tasks. When new data arrives, existing prior art of FM mainly focuses on retraining compressed models of predetermined network architectures, limiting the feasibility of edge devices to efficiently achieve high accuracy for FMs. In this paper, we propose EdgeTA, a neuron-grained FM scaling system to maximize the overall accuracy of FMs promptly in response to their data dynamics. EdgeTA's key design features in scaling are (i) proxy mechanism, which adaptively transforms a FM into a compact architecture retaining the most important neurons to the input data, and (ii) neuron-grained scheduler, which jointly optimizes model sizes and resource allocation for all FMs on edge devices. Under tight retraining window and limited device resources, the design of EdgeTA can achieve most of the original FM's accuracy with much smaller retraining costs. We implement EdgeTA on FMs of natural language processing, computer vision and multimodal applications. Comparison results against state-of-the-art techniques show that our approach improves accuracy by 21.88% and reduces memory footprint and energy consumptions by 27.14% and 65.65%, while further achieving 15.96% overall accuracy improvement via neuron-grained scheduling.**

***Index Terms*—Evolving data, foundation model, neuron-grained scaling, resource scheduling, retraining.**

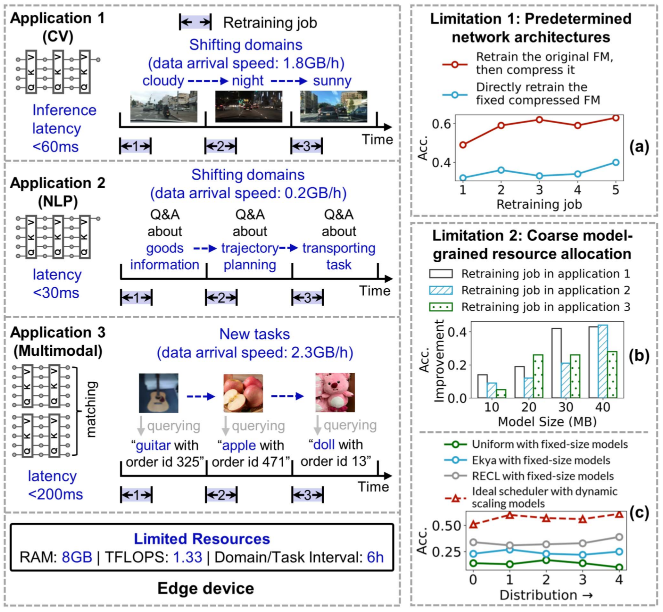

Application 1 (CV)
Retraining job
Shifting domains (data arrival speed: 1.8GB/h)
cloudy → night → sunny
Inference latency <60ms
1 2 3
Time
Application 2 (NLP)
Shifting domains (data arrival speed: 0.2GB/h)
Q&A about goods information → Q&A about trajectory planning → Q&A about transporting task
latency <30ms
Application 3 (Multimodal)
New tasks (data arrival speed: 2.3GB/h)
matching
querying "guitar with order id 325"
querying "apple with order id 471"
querying "doll with order id 13"
latency <200ms
Limited Resources
RAM: 8GB | TFLOPS: 1.33 | Domain/Task Interval: 6h
Edge device
Limitation 1: Predetermined network architectures
Retrain the original FM, then compress it
Directly retrain the fixed compressed FM
Acc.
Retraining job
(a)
Limitation 2: Coarse model-grained resource allocation
Retraining job in application 1
Retraining job in application 2
Retraining job in application 3
Acc. Improvement
Model Size (MB)
(b)
Uniform with fixed-size models
Ekya with fixed-size models
RECL with fixed-size models
Ideal scheduler with dynamic scaling models
Acc.
Distribution →
(c)

Fig. 1. An example scenario of edge-based model retraining.

I. Introduction

FOUNDATION models (FMs), e.g. BERT and GPT, are becoming standard building blocks for artificial intelligence (AI) systems [9]. Such large-scale and pre-trained FMs are also increasingly adopted on AI applications at edge, ranging from natural language processing (NLP) [77] and computer vision (CV) [66] to multimodal applications [75]. FMs outperform traditional application-tailored models in both accuracy [32], [78], [78], [92] and sample efficiency (requiring fewer samples to reach the same performance) [49]. To deploy a FM on an edge device, existing adaptation techniques need to retrain/fine-tune it according to an application's specific input data [77], [79], [83], [86], [98], [103], and compress it to meet the stringent inference latency constraints [40], [75]. Such compressed models encounter the challenge of evolving input data at edge and hence need continuously retraining to maintain consistent accuracy [6], [51]. The state-of-the-art [11], [14] takes up to hours to retrain such FMs on powerful GPU servers or multiple GPU edge devices [4], [55], [58], [68], [69], thus calling for new and lightweight methodologies especially for edge devices.

Fig. 1 illustrates an example scenario where an unmanned car runs three concurrent FMs for autonomous driving, interaction with humans, and goods querying applications, respectively. This scenario poses two key challenges in edge-side FM retraining: (1) *evolving data*: each application faces continuously evolving input distributions consisting of new/unseen *domains* (e.g. unlabeled distribution shifts in applications 1 and 2) or *tasks* (e.g. new classes in application 3's labeled data), and they

Received 28 February 2024; revised 9 November 2024; accepted 11 November 2024. Date of publication 22 November 2024; date of current version 6 March 2025. This work was supported in part by the National Key Research and Development Program of China under Grant 2023YFE0209100, in part by the National Natural Science Foundation of China under Grant 62272046, Grant 62132019, and Grant 61872337, and in part by the high-quality development project of the Ministry of Industry and Information Technology under Grant CEIEC-20240. Recommended for acceptance by H. Seferoglu. *(Corresponding author: Rui Han.)*

Qinglong Zhang, Rui Han, Chi Harold Liu, and Guoren Wang are with the Beijing Institute of Technology, Beijing 100811, China (e-mail: 3120211050@bit.edu.cn; hanrui@bit.edu.cn; chiliu@bit.edu.cn; wanggr@bit.edu.cn).

Song Guo is with the Hong Kong University of Science and Technology, Hong Kong (e-mail: songguo@cse.ust.hk).

Lydia Y. Chen is with TU Delft, 2628 CD Delft, Netherlands (e-mail: lydiaychen@ieee.org).

Digital Object Identifier 10.1109/TMC.2024.3504859

arrive in high volume (0.2 to 2.3 GB/second). (2) *Limited computational resources* (GPU cycles): the device has small GPU cores (TFLOPS is 1.33) and memory (8 GB), which are shared by multiple retraining jobs and other programs. Moreover, each retraining job needs to be completed within a short retraining window (e.g. 10 minutes) to maintain consistent accuracy of applications [6]. The success of FMs is mainly credited to their large *model sizes*, diverse *training data*, and massive amounts of *computational resources*– key factors of neural scaling law [29], [82]. However, small/compressed model size and tight resource budget greatly increase the difficulty of effective learning on challenging patterns of evolving data.

Predetermined network architectures lower model accuracy in learning new data: On edge devices, it is infeasible to retrain original FMs even using the fastest adaptation techniques such as Parameter-Efficient Fine-Tuning (PEFT) [37]. This is because although they only update a small proportion of model parameters in retraining, they still need to perform a forward operation on all parameters at each iteration and this operation may exceed the memory limit. For instance, LoRA [41] consumes 53.1 GB memory in forward when retraining GPT-Neo with batch size 16. Hence current edge retraining systems [6], [51] directly retrain compressed FMs that have restricted learning capacity when learning new tasks [3], [13], [47], [54], [73] or domains [8], [27], [59], [70], [87], [96], [97], [100]. In Fig. 1(a)'s example with five retraining jobs, directly retraining the compressed FM results in 23.40% lower accuracy than the computationally expensive method that first retrains the original model and then compresses it. This is because in each retaining job, the later method dynamically changes the network architecture of the compressed model such that it retains the most important parameters to the newly arrived data. The *first challenge*, therefore, is *how to keep the small model size for low-overhead on-device retraining, while dynamically adapting network structures to evolving input data for high accuracy.*

Coarse model granularity hinders optimizing resource allocation: Under limited device resources and short retraining windows, how to allocate resources among co-running retraining jobs is critical for model performance. Current edge schedulers for inference jobs make the latency-accuracy tradeoff using model upsizing/downsizing and offline profiling, under the strong assumption of data input is stationary [5], [28], [34], [46], [65]. In contrast, schedulers (Uniform, Ekya [6] and RECL [51]) for retraining jobs focus on allocating resources to maximize their overall accuracy, using fixed-size compressed models. However, allocating the same amount of resource to a model may result into considerably different accuracy improvements when handling different retraining jobs, as shown in Fig. 1(b). The ideal scheduler, which scales up models with higher accuracy improvements and allocates more resources to them (and vice verse in scaling down models), hence can achieve much higher accuracies than the current edge schedulers (Fig. 1(c)). The *second challenge* now requires us to investigate *how to manage resource allocation at a finer model granularity*.

In this paper, we design EdgeTA, an Edge-side Transformer Adaptation system to maximize the overall accuracy of multiple FMs in response to their data dynamics at run-time. The key idea of EdgeTA is to perform FMs' retraining at the granularity of *neuron* [17], [81], the basic computational unit that forms FMs'/transformers' layers. For each retraining job, EdgeTA identifies a small proportion of neurons from its FM that are most important to the current input data's accuracy and uses them to construct a *proxy model* at run-time. EdgeTA also constructs neuron indexes to record these neurons' mapping relationship between the proxy model and the FM, thus supporting their low-cost collaborative training via the proxy model. In multi-application scenario, EdgeTA supports the dynamic scaling of their FMs and thus enables neuron-grained resource scheduling to optimize the overall accuracy of multiple retraining jobs. In particular, the contributions of this paper are as follows:

- *Fast and accurate adaptation via proxy model:* EdgeTA introduces a novel concept of *proxy model* that extracts a FM's neurons most important to the arriving input distribution, thus training this model to achieve both high accuracy and low cost for adaptation. To maintain model generality, EdgeTA also employs a *knowledge base* to accumulate the learned knowledge and transfer this knowledge to the original FM with small overheads.
- *Neuron-grained resource scheduling with dynamic model scaling:* EdgeTA supports dynamic model scaling at the neuron granularity and online estimation of different model sizes' accuracy improvements per resource unit usage, thereby achieving fine-grained resource scheduling by first allocating resources to the model neurons with the largest accuracy improvement.
- *Lightweight system design and applicability on variety of FMs:* EdgeTA implements lightweight systems to perform online scaling, accuracy improvement estimation and retraining of FMs with low overheads. Our approach is designed to support prevalent edge-side applications of different modalities (image, text and multimodal data) and its implementation supports both Hugging face transformers and other user-specified FMs, and it is available at https://huggingface.co/spaces/LINC-BIT/EdgeTA.

Summary of experimental results: We evaluate EdgeTA on four commodity edge devices whose GPU ranges from 256 cores to 1972 cores and memory size ranges from 8 GB to 32 GB. Using six representative workloads in CV, NLP and multimodal applications, our experiments compare against the state-of-the-art FM adaptation techniques in both single- and multi-application scenarios: (i) *evolving input data adaptation*. We compare EdgeTA against 13 input data adaptation methods and the results show: under the same resource budgets, our approach improves accuracy by 21.88% in average (up to 35.80%), while reducing memory footprint and energy consumptions by 27.14% and 65.65% respectively. (ii) *Multi-application scenarios*. Compared to five edge schedulers, our neuron-grained resource scheduler achieves 15.96% improvement in the overall accuracy of all retraining jobs. EdgeTA can also be integrated with typical server-based resource schedulers to improve model accuracy by 14.23%.

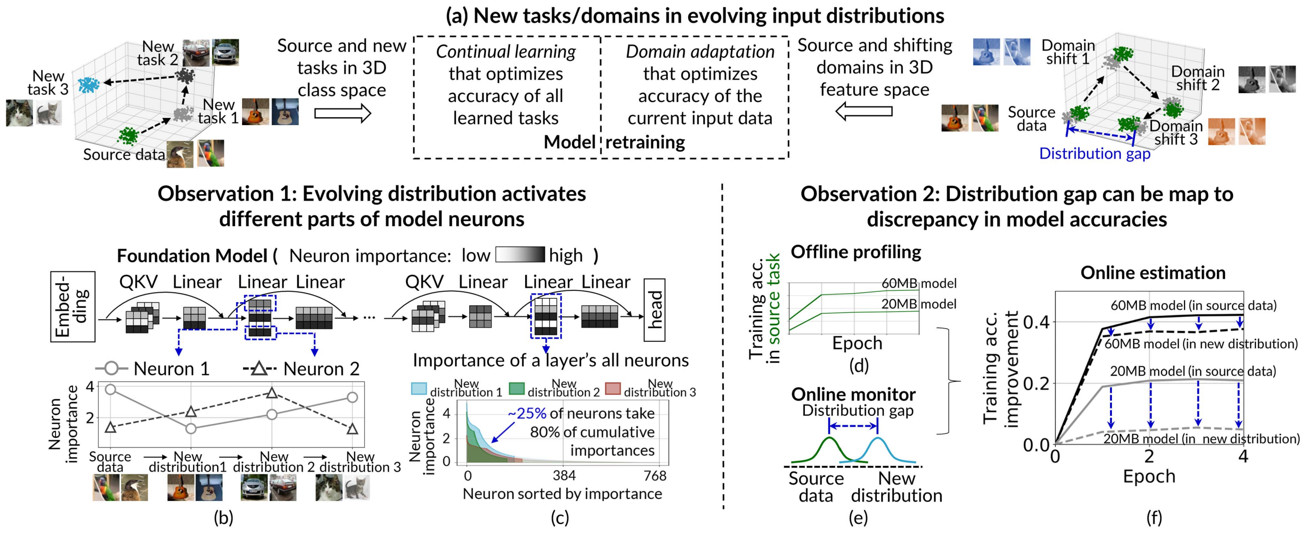

(a) New tasks/domains in evolving input distributions
Source and new tasks in 3D class space
Continual learning that optimizes accuracy of all learned tasks
Domain adaptation that optimizes accuracy of the current input data
Model retraining
Source and shifting domains in 3D feature space
Observation 1: Evolving distribution activates different parts of model neurons
Observation 2: Distribution gap can be map to discrepancy in model accuracies
Foundation Model (Neuron importance: low high)
Offline profiling
Online estimation
Online monitor
Distribution gap
Importance of a layer's all neurons
~25% of neurons take 80% of cumulative importances
Neuron sorted by importance
(b)
(c)
(d)
(e)
(f)

Fig. 2. A motivation example to understand EdgeTA.

II. Motivation

In this section, we lay out two observations to understand how evolving data and model retraining requires FM scaling, and present the design objectives of EdgeTA.

New tasks/domain shifts in evolving input data: A pre-trained FM is a general purpose model for diverse input distribution space. When adapting to an application, it is fine tuned according to its source data offline, as illustrated in Fig. 2(a). In dynamic edge environment, the deployed model usually encounters evolving input data consisting of new tasks and/or domain shifts and thus needs retraining. Specifically, *continual learning* methods update the model according to new tasks that have multiple classes (e.g. different cats or vehicles) not seen in the source data [105], and maximize the overall accuracy of all tasks. *Domain adaptation* methods address domain shifts in input features (e.g. light illumination or background) to mitigate the model's accuracy depredation. In both types of model retraining, we have two observations as follows.

Observation 1. Evolving input data activates different model neurons: The intrinsic property of neural networks means a FM's neurons have differentiated *importance values* to different input distributions [36] (Fig. 2(b)). During the forward operation, a neuron's *importance value* represents the amount of input data information its contains. This value can be calculated using Feature Boosting and Suppression (FBS) [30] or Squeeze-and-Excitation (SE) module [42]. When further considering all neurons in a layer, Fig. 2(c) shows they have a long tail such that most of the neurons have much smaller importance values than a small percentage of neurons with large values. However, previous studies only use neuron importance values in offline structured model pruning [53], [57] or online model scaling in inference jobs [28], [33], based on a strong assumption of stationary input distribution.

Design objective 1: EdgeTA adaptively transforms the FM into a small model that retains the FM's most important neurons for the newly arrived data, thus achieving both low overheads and high accuracy in retraining.

Observation 2. Distribution distance can be mapped to discrepancy in model accuracies: When scheduling resources (GPU cycles) among retraining jobs of variable model sizes, profiling these jobs' accuracy improvement per resource unit usage is a necessary condition. However, each profiles needs at least several training iterations and hence it is computationally expensive on edge devices. Previous studies show that it is possible to perform such profiling for the source data offline [28], [33] (Fig. 2(d)) and, at the same time, calculating the distribution distance between the source data and the newly arrived data can be completed quickly online (Fig. 2(e)). For example, the Frechet distance [26] calculates the distance between two distributions using their average feature vectors μ and covariance matrices. Hence it is possible to calibrate a FM's accuracy improvement for a new distribution according to its distance to the source data (Fig. 2(f)).

Design objective 2: EdgeTA develops an online estimator to accurately predict accuracy improvements for models of variable sizes, thus supporting neuron-grained resource scheduling of retraining jobs.

III. Design of EdgeTA

A. Overview

EdgeTA is designed based on the network neurons which widely exist in today's FMs/transformers. We note that FMs differ from traditional CNNs from two aspects: (1) FMs are pre-trained with diverse data and its neurons store more generic knowledge than filters (the basic computational unit in CNNs). Hence before deployment, a FM needs to be fine tuned to adapt to an application's specific input data (2) The number of neurons in a FM is usually much larger than the number of filters in a CNN (e.g. 50x larger), and the proportions of model parameters in

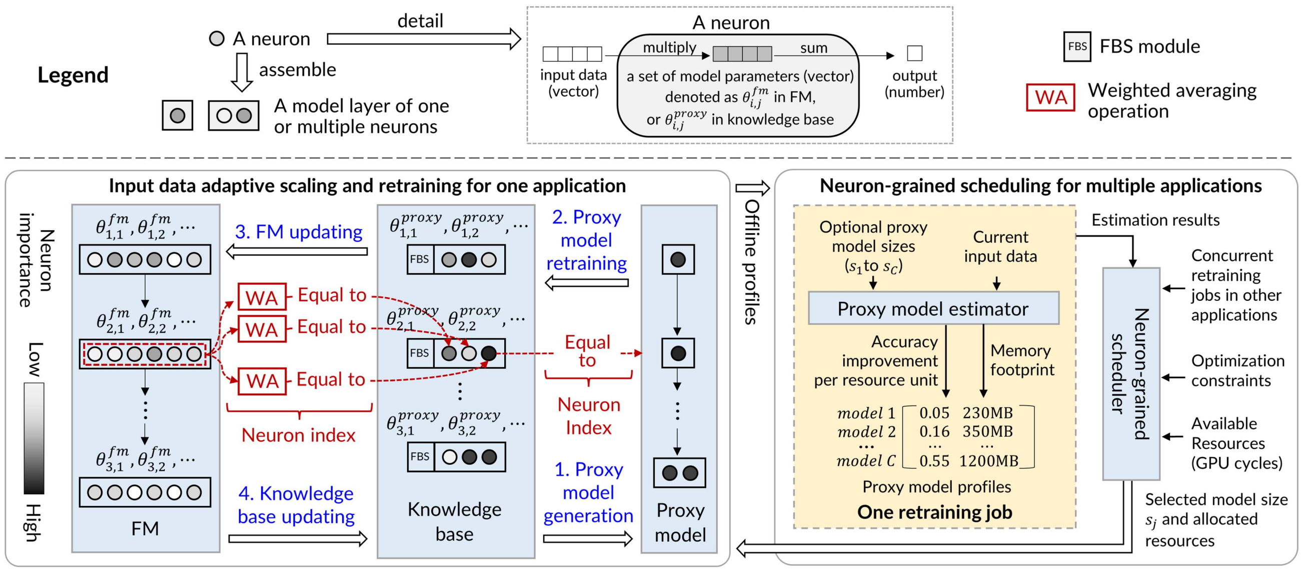

Legend
A neuron
detail
assemble
A model layer of one or multiple neurons
A neuron
input data (vector)
multiply
sum
a set of model parameters (vector) denoted as $\theta^{fm}_{i,j}$ in FM, or $\theta^{proxy}_{i,j}$ in knowledge base
output (number)
FBS module
WA Weighted averaging operation
Input data adaptive scaling and retraining for one application
Neuron importance
Low
High
FM
3. FM updating
4. Knowledge base updating
Equal to
Neuron index
Knowledge base
2. Proxy model retraining
1. Proxy model generation
Neuron Index
Proxy model
Offline profiles
Neuron-grained scheduling for multiple applications
Optional proxy model sizes (s_1 to s_C)
Current input data
Proxy model estimator
Accuracy improvement per resource unit
Memory footprint
model 1 0.05 230MB
model 2 0.16 350MB
model C 0.55 1200MB
Proxy model profiles
One retraining job
Estimation results
Neuron-grained scheduler
Concurrent retraining jobs in other applications
Optimization constraints
Available Resources (GPU cycles)
Selected model size s_j and allocated resources

Fig. 3. EdgeTA architecture.

neurons of different network layers are highly unbalanced [17], [81]. For instance, the linear layers (including QKV, attention, and feed forward layers) have about 30% of neurons in the model and these neurons contain over 99% of model parameters. Hence the retraining job needs to carefully select neurons to process to avoid high costs. Based on the characteristics of neurons, EdgeTA designs proxy mechanism and neuron index to support dynamic scaling of FMs in response to evolving input data at runtime.

First, EdgeTA's proxy mechanism adapts a generic large FM to a distribution-specific small model using two stages. At the offline stage, it transforms the FM into a much smaller and application-specific *knowledge base*, namely multiple network layers of neurons, which supports the input-adaptive and fast generation of small proxy models in retraining. At the online stage, once a retraining job starts, it constructs a *proxy model* that retains a small proportion (e.g. 20%) of knowledge base's most important neurons to the current input data. This model only retains the distribution-specific knowledge in the FM's neurons, thus consuming much smaller training resources than the original FM while achieving most of its model accuracy. We note that this proxy mechanism also alleviates catastrophic forgetting in model continuous model retraining [18]. This is because the retraining of the proxy model only updates the neurons most relevant to the current input data, and maintains other neurons of the FM unchanged. After retraining, the proxy model is used during inference until the next input distribution shift happens.

Second, EdgeTA develops *neuron indexes* to support the dynamic model scaling at the neuron granularity with two objectives: (1) the neuron indexes leverage the dynamic importance values of neurons and hence can achieve accuracy-size tradeoff in a finer granularity. (2) The number of neurons is much smaller (e.g. 1000x smaller) than that of model parameters, thus guaranteeing the small memory footprint of neuron indexes and fast processing of neurons in model retraining. Given that most model parameters are stored in linear layers' neurons, EdgeTA only constructs neuron indexes for linear layers to further reduce the overheads. With neuron indexes, neurons' mapping relationships between the FM and the knowledge base, and between the knowledge base and the proxy model can be recorded for quick knowledge transfer. This step accumulates the proxy model's learned knowledge in the knowledge based and the FM, and can mitigate catastrophic forgetting in continuous model retraining.

Fig. 3 illustrates the architecture of EdgeTA, which is split into single-application and multi-application parts. EdgeTA operates at the neuron granularity. A *neuron* consists of a set of model parameters, and it takes a vector as input and outputs a number. A model layer usually contains hundreds of neurons.

Neuron-grained FM scaling and retraining for one application (Section III-B). In EdgeTA, the knowledge base and its corresponding neuron index are first constructed offline and then used to support online scaling of the application's FM. Specifically, EdgeTA first takes the pre-trained FM (its i-th layer's j-th neuron is denoted as $\theta^{fm}_{i,j}$) and the source data of an application as inputs, and outputs the application-specific knowledge base (that is, a matrix of neurons, in which the i-th layer's j-th neuron is denoted as $\theta^{proxy}_{i,j}$), its neuron indexes to the FM, and profiles of the FM's accuracy improvements under different model sizes (Section III-B1). In a neuron index, a weighted average of multiple neurons in the FM corresponds to a neuron in the knowledge base. The construction process is only executed once offline.

At the online stage, EdgeTA first sends offline profiles to the neuron-grained scheduler and obtains the selected model size and allocated resources (i.e. GPU cycles). It then performs FM scaling and retraining with four steps. At step 1, the *distribution-adaptive model generator* (Section III-B2) first extracts a *proxy model* from the knowledge base to retain the FM's most

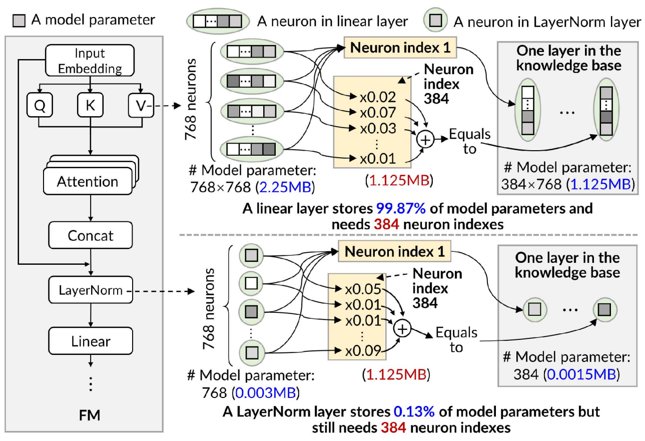

A model parameter
A neuron in linear layer
A neuron in LayerNorm layer
Input Embedding
Q
K
V
Attention
Concat
LayerNorm
Linear
FM
768 neurons
Neuron index 1
Neuron index 384
x0.02
x0.07
x0.03
x0.01
Equals to
One layer in the knowledge base
# Model parameter: 768×768 (2.25MB)
(1.125MB)
# Model parameter: 384×768 (1.125MB)
A linear layer stores 99.87% of model parameters and needs 384 neuron indexes
768 neurons
Neuron index 1
Neuron index 384
x0.05
x0.01
x0.01
x0.09
Equals to
One layer in the knowledge base
# Model parameter: 768 (0.003MB)
(1.125MB)
# Model parameter: 384 (0.0015MB)
A LayerNorm layer stores 0.13% of model parameters but still needs 384 neuron indexes

Fig. 4. Example neuron indexes in a transformer.

important neurons (i.e. the neurons with the darkest colors in Fig. 3) to the current distribution, and then constructs this model's neuron indexes to the knowledge base. Each neuron index records one and the same neuron in the proxy model and the knowledge base. In the following three steps, the *collaborative learner of FM and proxy model* (Section III-B3) sequentially retrains the proxy model (step 2), updates the FM (step 3), and updates the knowledge base (step 4). The last two steps only need quick tensor addition and multiplication operations (rather than slow iterative training operations), and they aim to accumulate the proxy model's learned knowledge in the FM and knowledge base.

Neuron-grained resource scheduling among multiple applications (Section III-C). The scheduling consists of two modules: the *proxy model estimator* (Section III-C1) takes optional proxy model sizes (s_1 to s_C) and the current input distribution as inputs, and calculates this model's accuracy improvement per resource unit and memory footprint for each model size. The *neuron-grained scheduler* (Section III-C2) checks up all retraining jobs' estimation results and the edge system's available resources, and finds the optimal model sizes and assigned resources for these jobs that maximize their overall accuracy under the optimization constraints (e.g. device memory capacity).

B. Neuron-Grained FM Scaling and Retraining

1) Knowledge Base and Neuron Index: To support online scaling of a FM at the neuron granularity, EdgeTA first constructs knowledge base and its neuron indexes offline, as illustrated in Fig. 4.

First, for the same set of neurons in a layer of the FM, EdgeTA constructs multiple neuron indexes and thus retains multiple neurons in the *knowledge base*. For example, in Fig. 4's FM, each layer has 768 neurons (each neuron consists of 768 model parameters). Using one neuron index, these neurons are combined into one neuron in the knowledge base. Overall, 384 neurons are constructed in one layer of the knowledge base using 384 different neuron indexes. These indexes represent different ways of combining the neurons in the FM, thereby increasing the diversity of the knowledge base and improving its learning

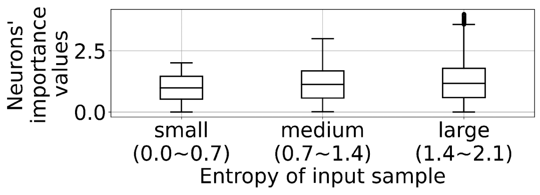


Fig. 5. The correlation between the input sample's entropy and the distribution of neurons' importance values.

capacity for unseen distributions. Formally, the knowledge base is defined as a two-dimensional matrix θ^P of neurons:

$$\begin{bmatrix} \theta^{proxy}_{1,1} & \theta^{proxy}_{1,2} & \cdots & FBS \\ \vdots & \vdots & \vdots & \\ \theta^{proxy}_{n,1} & \theta^{proxy}_{n,2} & \cdots & FBS \end{bmatrix} \tag{1}$$

where $\theta^{proxy}_{i,j}$ represents the i-th layer's j-th neuron in the knowledge base, and n represents the number of layers. Each line represents the neurons in one layer, and it also contains a Feature Boosting and Suppression (FBS) module [30], which is used to identify the most important neurons in proxy model generation.

Second, a *neuron index* records how a layer of neurons in the FM are combined into one neuron in the knowledge base. Formally, let $\theta^{fm}_{i,1}, \theta^{fm}_{i,2}, \ldots$ be the neurons in the FM's i-th layer and $\theta^{proxy}_{i,j}$ be one neuron in the knowledge base, a neuron index is a set of relevance coefficients $\{\Gamma_{i,j,1}, \Gamma_{i,j,2}, \ldots\}$:

$$\theta^{proxy}_{i,j} = \sum_k \theta^{fm}_{i,k} \cdot \Gamma_{i,j,k} \tag{2}$$

where $\Gamma_{i,j,k}$ is the relevance coefficient between $\theta^{proxy}_{i,j}$ and $\theta^{fm}_{i,k}$, and k ranges from 1 to the number of neurons in the FM's i-th layer. In Fig. 4's example, EdgeTA only constructs neuron indexes for linear layers because they contain 99.87% of model parameters but have the same number (384) of neuron indexes as the LayerNorm layer.

2) Distribution-Adaptive Model Generator: Given a new input distribution and a selected model size (by the neuron-grained scheduler), the generator transforms the FM into a distribution-adaptive proxy model and constructs its neuron indexes to the knowledge base with three steps.

Representative sample selection: Given v samples $\vec{x}_1$ to $\vec{x}_v$, step 1 selects the one with the highest information entropy [59]: $\vec{x}^* = \arg\min_{\vec{x}_i \in \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_v\}} -\sum_c p(y^c_i) \cdot log(p(y^c_i))$, where $p(y^c_i)$ represents the probability that $\vec{x}_i$ is predicted as class c. The selected sample denotes the most difficult one in the input data and it provides the largest amount of information about how to learn features for correct classification [71]. In other words, this sample maximizes the difference among neurons' importance values. Taking ViT-B as an example, Fig. 5 visualizes the correlation between the input sample's entropy (small, medium, and large) and the distribution of neurons' importance values. We can see the sample with the highest entropy (e.g. the most difficult sample) leads to the largest difference of neurons' importance values.

Proxy model generation: Step 2 first uses the model neurons and the FBS module in the knowledge base to generate an intermediate model using multiple iterations. Each iteration constructs a model layer by combining one layer of the knowledge base, a parallel network branch, and a FBS module. Subsequently, this step identifies the neurons with the highest importances from the intermediate model, and assembles them into a proxy model. Formally, given a neuron $\theta_{i,j}^{proxy}$, its *neuron importance* $\mathcal{Q}(\theta_{i,j}^{proxy})$ is defined as the amount of input information by taking sample $\vec{x}^*$ as input:

$$\vec{p}_i = FBS(\mathcal{F}_i),\ \mathcal{Q}(\theta_{i,j}^{proxy}) = \vec{p}_{i,j} \tag{3}$$

$$FBS(\mathcal{F}_i) = ReLU(Linear(AvgPool(\mathcal{F}_i))) \tag{4}$$

$$\mathcal{F}_1 = \vec{x}_j^* \tag{5}$$

where $\mathcal{F}_i$ is the input of the knowledge base's i-th layer. In (3), the FBS module takes $\mathcal{F}_i$ as input, and outputs a vector $\vec{p}_i$ whose j-th element $\vec{p}_{i,j}$ is $\theta_{i,j}^{proxy}$'s neuron importance. (4) represents that a FBS module consists of one average pooling layer, one linear layer, and one ReLU layer. (5) represents that $\vec{x}^*$ is the input of the knowledge base's first layer.

Note that our approach utilizes the FBS module to estimate neurons' importance. This is because this module is used at the inference stage in traditional approaches and it only needs one forward operation in estimation. In contrast, current model pruning techniques [15] either calculate neurons independent of input distributions (data-free techniques), or require at least a batch of input samples to calculate importance and are much slower.

Comparative analysis with hot/cold neurons: PowerInfer [81] is a novel method that employs a neuron's activation frequency to measure its importance, and thus divides neurons into hot or cold ones according to their frequencies. There exists two differences between PowerInfer and EdgeTA. First, the proportions of hot and cold neurons (e.g. 20% and 80%) are predetermined at the offline pre-training stage, because PowerInfer is developed for model inference and the input distribution is stationary. In contrast, EdgtTA is developed for evolving input distributions. Each time a retraining is triggered, neurons are first ranked according to their importances to the current input data. The proportion of neurons retrained in the proxy model is then determined online according to the allocated resources to this retraining. Second, in PowerInfer, hot and cold neurons are loaded into Video RAM (VRAM) and RAM and processed by GPU and CPU in inference, respectively. In EdgeTA, only the neurons in the proxy model are loaded into VRAM and processed in retraining.

Neuron index construction: For each selected neuron, step 3 constructs a neuron index to record its mapping relationship to the neuron in the knowledge base. This index guarantees that the model parameters of two connected neurons should keep the same.

3) Collaborative Learner of FM and Proxy Model: The collaborative learner first retrains the proxy model to provide high accuracy for the current input data, then updates the FM to accumulate the learned knowledge. Finally, it updates the knowledge base to transfer the latest FM's learning capacity to it.

Proxy model retraining: The learner only trains a small portion of neurons because other neurons have small gradients and skipping them can accelerate retraining with negligible accuracy losses [10]. To identify these neurons, a naive approach requires a full and expensive forward/backward to compute the gradients of all model parameters. In EdgeTA, the learner only calculates the gradients of parameters in the *normalization* layers. This is because in a typical transformer, each neuron in the *linear* layer pairs with a neuron in the *normalization* layer (LayerNorm or RMSNorm). That is, the gradients of model parameters in each paired neuron are positively correlated (e.g. the average Kendall's Tau correlation [50] is up to 0.858in ViT-B). However, the calculation of gradients is much more expensive in the linear layer. Hence our learner chooses to only calculate the parameters' gradients in the normalization layer and use these gradients to select neurons in both types of layers. Finally, the learner selects a small subset (e.g. 20%) of neurons that take most of (e.g. 80%) cumulative gradients, and updates these neurons using supervised/unsupervised retraining algorithms. The retraining can be further accelerated by distributing the computation to more edge devices [4], [55], [58], [68], [69].

FM updating: Let $\Delta\theta_{i,j}^{proxy}$ be the updating extent of neuron $\theta_{i,j}^{proxy}$ in the proxy model, the learner first updates this neuron's corresponding neuron in the knowledge base:

$$\theta_{i,j}^{proxy} \leftarrow \theta_{i,j}^{proxy} + \Delta\theta_{i,j}^{proxy} \tag{6}$$

where $\leftarrow$ denotes the assignment operation. Subsequently, the learner updates the FM according to the relevance coefficient $\Gamma_{i,j,k}$ between it and the knowledge base:

$$\theta_{i,k}^{fm} \leftarrow \theta_{i,k}^{fm} + \Gamma_{i,j,k} \cdot \Delta\theta_{i,j}^{proxy} \tag{7}$$

Note that the value of $\Gamma_{i,j,k}$ is set during the construction of neuron index offline, and represents the similarity of model parameters between the neuron $\theta_{i,k}^{fm}$ in the FM and the neuron $\theta_{i,j}^{proxy}$ in the proxy model. That is, two similar parameters also have similar updating directions in their loss space [22]. Hence when updating neuron $\theta_{i,k}^{fm}$ using (7), a large relevance coefficient means the proxy model can transfer more of its learned knowledge to the FM; otherwise, neuron $\theta_{i,k}^{fm}$ keeps more of its original knowledge.

Knowledge base updating: Based on the updated FM, the learner updates each neuron of the knowledge base as:

$$\theta_{i,j}^{proxy} \leftarrow \sum_k \theta_{i,k}^{fm} \cdot \Gamma_{i,j,k} \tag{8}$$

Overhead analysis: Without loss of generality, suppose the proxy model have ρn linear layers and $(1-\rho)n$ other layers $(0 < \rho < 1)$, whose model parameters have the shape $d \times d$ and $d \times 1$, respectively.

- *Gradient calculation:* The FLOPs and memory footprint are $(\rho n d^3 + (1-\rho)nd^2)$ and $\rho n d^2 + 2(1-\rho)nd$ for a full backward, and $(1-\rho)nd^2$ and $2(1-\rho)nd$ when only considering the normalization layers (saving one magnitude of FLOPs and memory).

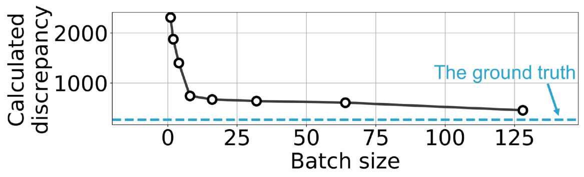


Fig. 6. Distribution discrepancies under different batch sizes.

- *Proxy model retraining:* The memory overhead is $16(\rho nd^2 + 2(1-\rho)nd)$ when training all neurons [74], and $(4+12\alpha\%)(\rho nd^2 + 2(1-\rho)nd)$ when only training $\alpha\%$ of neurons (saving $4/(1+3\alpha\%)$x memory, that is, 2.5x when $\alpha\% = 20\%$).
- *Feedback learning:* The FM and the knowledge base can be loaded into memory layer-by-layer. Assume they have 16x and 4x model parameters than the proxy model, the peak memory usage will be $22d$ (far smaller than an inference) and the FLOPs will be $68\rho nd^3$ (similar to an inference).

C. Neuron-Grained Scheduling Among Multiple Applications

Given A co-running applications, C optional model sizes $\{s_1, s_2, \ldots, s_C\}$ and a resource unit, suppose each application encounters its k-th input distribution, the *proxy model estimator* first predicts each application's accuracy improvement per resource unit for the current input distribution. The *neuron-grained resource scheduler* then takes the online estimation results, the offline profiles and the available resources (that is, retraining window t^{max} and available memory m^{max}) as inputs, and outputs all applications' model sizes and assigned resources to maximize their overall accuracy.

1) Proxy Model Estimator: The estimator takes the i-th application's accuracies before and after training in the source data (obtained offline), the discrepancy between the source data and the k-th newly arrived distribution, and a model size s_j as inputs, and outputs this application's accuracy before retraining $\mathcal{I}_{i,j}^{(k)}$, the accuracy after retraining $\tilde{\mathcal{I}}_{i,j}^{(k)}$, and the memory footprint $M_{i,j}$ $(1 \le i \le A, 1 \le j \le C)$. The estimation can be completed quickly for two reasons. First, it is based on a tiny neural network, which maps the discrepancy of two input distributions to the discrepancy in accuracies in retraining. Second, it only uses a small batch of samples to calculate the discrepancy of the current input distribution to the source data. Note that the batch size used to calculate this discrepancy may vary across different source datasets, because they have distinct feature space. Hence at the offline stage, the estimator predetermines this batch size using two steps. Step 1 applies data augmentation techniques in source data to simulate a newly arrived distribution. Step 2 gradually doubles the batch size (e.g. $8, 16, 32, \ldots$), and calculates the discrepancy between the simulated input distribution and the source data for each size. This step stops if the tested batch size brings a sufficient precision in discrepancy calculation: its calculated discrepancy has an error smaller than 10% compared to the ground truth (the discrepancy calculated using all samples). Taking the GTA5 [76] dataset as an example, Fig. 6 shows that a batch size of 128 leads to only 9.7% error to the ground truth.

Moreover, the estimator calibrates the k-th input distribution's prediction result using the *actual/monitored* accuracies of the $(k-1)$ seen distributions:

$$\mathcal{I}_{i,j}^{(k)} \leftarrow \frac{\mathcal{I}_{i,j}^{(k)} + \sum_{a=1}^{k-1} w_a \cdot \mathcal{I}_{i,j}^{(a)}}{2} \tag{9}$$

$$\tilde{\mathcal{I}}_{i,j}^{(k)} \leftarrow \frac{\tilde{\mathcal{I}}_{i,j}^{(k)} + \sum_{a=1}^{k-1} w_a \cdot \tilde{\mathcal{I}}_{i,j}^{(a)}}{2} \tag{10}$$

where $\mathcal{I}_{i,j}^{(a)}$ and $\tilde{\mathcal{I}}_{i,j}^{(a)}$ is the actual accuracies of the a-th distribution and w_a $(0 < a < k-1,\ 0 < w_a < 1)$ is the reciprocal of this distribution's discrepancy to the newly arrived distribution (normalized by softmax). In calibration, the estimator calculates the k-th distribution's accuracy by averaging it with the weighted combination of the $(k-1)$ previous distributions' accuracies (9) and (10).

In addition, the memory footprint $M_{i,j}$ is proportional to its size and it is estimated as (s_j / FM's model size $\times$ FM's memory footprint).

2) Neuron-Grained Scheduler: The scheduler aims to maximize the overall accuracy of all applications given a resource budget. To this end, it define the optimization objective as the accuracy improvement per unit of resource usage ((11)): for each application, this metric is the cumulative accuracy over the retraining window $\mathcal{I}_{i,j}^{(k)} \cdot \frac{T_{i,j}}{\mu_i^{(k)}} + \tilde{\mathcal{I}}_{i,j}^{(k)} \cdot (t^{max} - \frac{T_{i,j}}{\mu_i^{(k)}})$ divided by the retraining window t^{max}. With two constraints (retraining window t^{max} in (14) and available memory m^{max} in (15)), the optimization problem is formalised as follows:

$$\max_{S_{i,j}^{(k)}} \sum_{i=1}^{A} \sum_{j=1}^{C} \frac{\mathcal{I}_{i,j}^{(k)} \cdot \frac{T_{i,j}}{\mu_i^{(k)}} + \tilde{\mathcal{I}}_{i,j}^{(k)} \cdot \left(t^{max} - \frac{T_{i,j}}{\mu_i^{(k)}}\right)}{t^{max}} \cdot S_{i,j}^{(k)} \tag{11}$$

s.t.

$$S_{i,j}^{(k)} \in \{0,1\}, \ \sum_{j=1}^{C} S_{i,j}^{(k)} = 1 \tag{12}$$

$$\mu_i^{(k)} = \frac{\sum_{b=1}^{C} (\tilde{\mathcal{I}}_{i,b}^{(k)} - \mathcal{I}_{i,b}^{(k)}) \cdot S_{i,b}^{(k)}}{\sum_{a=1}^{A} \sum_{b=1}^{C} (\tilde{\mathcal{I}}_{a,b}^{(k)} - \mathcal{I}_{a,b}^{(k)}) \cdot S_{a,b}^{(k)}} \tag{13}$$

$$\forall\, 1 \le i \le A, \ \sum_{j=1}^{C} \frac{T_{i,j}}{\mu_i^{(k)}} \cdot S_{i,j}^{(k)} \le t^{max} \tag{14}$$

$$\sum_{i=1}^{A} \sum_{j=1}^{C} M_{i,j} \cdot S_{i,j}^{(k)} \le m^{max} \tag{15}$$

- $S_{i,j}^{(k)}$ indicates whether the i-th application chooses the j-th model size for the k-th input distribution ($S_{i,j}^{(k)} = 1$) or not ($S_{i,j}^{(k)} = 0$).
- $\mu_i^{(k)}$ is the proportion of allocated GPU cycles for the i-th application. It is equal to the ratio of this application's accuracy improvement $\sum_{b=1}^{C} (\tilde{\mathcal{I}}_{i,b}^{(k)} - \mathcal{I}_{i,b}^{(k)}) \cdot S_{i,b}^{(k)}$ to the sum of all applications' accuracy improvements. This

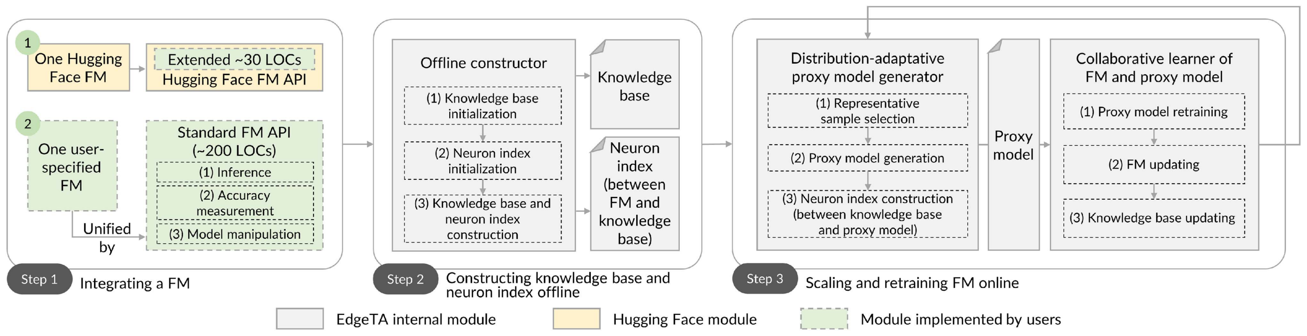

1
One Hugging Face FM
Extended ~30 LOCs Hugging Face FM API
2
One user-specified FM
Unified by
Standard FM API (~200 LOCs)
(1) Inference
(2) Accuracy measurement
(3) Model manipulation
Step 1 Integrating a FM
Offline constructor
(1) Knowledge base initialization
(2) Neuron index initialization
(3) Knowledge base and neuron index construction
Knowledge base
Neuron index (between FM and knowledge base)
Step 2 Constructing knowledge base and neuron index offline
Distribution-adaptative proxy model generator
(1) Representative sample selection
(2) Proxy model generation
(3) Neuron index construction (between knowledge base and proxy model)
Proxy model
Collaborative learner of FM and proxy model
(1) Proxy model retraining
(2) FM updating
(3) Knowledge base updating
Step 3 Scaling and retraining FM online
EdgeTA internal module
Hugging Face module
Module implemented by users

Fig. 7. System implementation of EdgeTA.

metric guides the scheduler to allocate more GPU cycles to applications with higher accuracy improvements.

- Given the total retraining window t^{max}, the i-th application's retraining time $T_{i,j}$ is amplified by $1/\mu_i^{(k)}$ times because its retraining job uses only $\mu_i^{(k)}$ of GPU cycles.
- The sum of all applications' memory footprint should be smaller than the device's available memory m^{max}.

The above optimization problem is non-linear and non-convex. The scheduler uses the genetic algorithm to search the best solution by repeating the three steps. Step 1 generates new solutions by initializing random solutions or mutating previously retained solutions. Step 2 computes the solutions' fitness (the value that each solution corresponds to in (11)). Finally, step 3 retains solutions with higher fitness. If it cannot find better solutions, it stops and outputs the currently found best solution. This search completes within a few seconds because the solution space is small ($A, C \leq 10$ in typical edge scenarios) and it can compute the fitness of multiple solutions in parallel.

IV. Implementation

EdgeTA is implemented in Python with 8k LOCs and it is currently targeted for transformers [91] running on commodity edge devices and Linux environment. Its scaling and retraining of transformers are implemented based on timm 0.9.1 and transformers 4.30.2. Its scheduler is built upon the optimization problem solver in scikit-opt 0.6.6 and resource management systems in Docker 10.03.6 and K3s 1.18.12.

Applicability: Fig. 7 illustrates the three steps of running a FM using EdgeTA. To facilitate the integration of a model, EdgeTA decouples the integration of a model (step 1) from its offline construction of knowledge base and neuron index (step 2) and online scaling and retraining of FM (step 3). This system design allows users only to implement the FM API at step 1 to integrate a model. Specifically, EdgeTA supports two types of models.

▷ *Hugging Face FMs [94]:* We implement EdgeTA to support FM APIs in the Hugging Face AI community. Using the `AutoModel` as example, EdgeTA calls function `AutoModel.from_pretrained()` to initialize a FM and calls function `AutoModel.forward()` to perform a forward operation. EdgeTA allows users to run a Hugging Face's FM using about 30 LOCs.

▷ *User-specified FMs:* EdgeTA designs a standard FM API (colored by green in Fig. 7) to unify user specified FM implementations. This API mainly defines: (i) how the FM performs an inference using the given sample; (ii) how the accuracy of the FM is measured using the given test dataset; (iii) how to manipulate (e.g. compress/update/remove) a specific layer in the FM. For each FM, this API can be implemented using about 200 LOCs.

Moreover, EdgeTA features on *lightweight system implementations* from two considerations. First, it addresses three major overheads in retraining: (1) each layer of the knowledge base is divided into multiple pieces in inferences to reduce the memory overhead; (2) when updating the FM and the knowledge base, its neuron indexes allow each network layer to be processed separately to reduce memory footprint. and (3) the *step*() function in PyTorch's optimizers is revised to freeze model parameters of small gradients to save computational costs. Moreover, it employs the thread pool to parallelize and accelerate the the optimization problem solving, and implements resource allocation based on the open source K3s [2] (a lightweight version of Kubernetes).

V. Evaluation

A. Settings

Testbeds: We choose four different edge devices imposing different architectural features: NVIDIA Jetson TX2, Xavier NX, AGX Xavier, and AGX Orin, whose processors are 256-core Pascal GPU, 384-core Volta GPU, 512-core Volta GPU, and 1792-core Ampere GPU, respectively; and memory sizes are 8 GB, 16 GB, 32 GB, and 32 GB, respectively. All devices are equipped with the operating system Ubuntu 18.04.5 and they run transformers using PyTorch 1.9.0.

FM models, datasets, and applications: To evaluate EdgeTA's generalization capability on different edge applications, we selected three most important applications, as listed in Table I.

- *CV application:* It takes images and videos as inputs. Without loss of generality, we select three popular workloads: image classification, semantic segmentation that recognizes the category of each pixel in an image, and object detection that recognizes the category and the location of all objects in an image.

TABLE I
SUMMARY OF APPLICATIONS AND WORKLOADS

Application: workload		**Source data**	**New arrival input data**	**FM**
CV: image classification	New tasks	GTA5 [76] Supervisely-Person [16]	MSCOCO2017 [60]	ViT-B/16 [25]
	Domain shifts		Cityscapes [16] BaiduPerson [95]	
CV: semantic segmentation	Domain shifts			
CV: object detection	Domain shifts			
NLP: text classification	New tasks	HL5Domains [43]	20Newsgroups [1]	Bert$_{base}$ [20]
	Domain shifts		Liu3Domains [61] Ding9Domains [23] SemEval14 [85]	
NLP: POS tagging	Domain shifts			
Multimodal: visual question answering	Domain shifts	VQAv2's first 1000 classes [31]	VQAv2-C [31], [39]	ViLT-B/32 [52]
	New tasks		VQAv2's last 2129 classes [31]	

- *NLP application:* It analyzes text data and we select two most commonly used workloads: text classification and part-of-speech (POS) tagging that categorizes a sentence's each word into a specific part of speech (e.g. noun and verb).
- *Multimodal application:* It analyzes text-image matches and we test the visual question answering workload that gives a text answer according to an image and the text question.

Accuracy metrics: For CV applications, top-1 classification accuracy, mIoU (mean Intersection over Union), and mAP@0.5 (mean average precision over Intersection over Union (IoU) threshold 0.5) are used to measure image classification, semantic segmentation, and object detection workloads, respectively. In addition, top-1 accuracy is used to measure two NLP workloads, and VQA score (the accuracy of the generated answers compared to the actual human answers) is to measure the multimodal workload.

B. Evaluation of FM Retraining Under Evolving Input Data

Here we compare EdgeTA with two types of baseline adaptation methods and also report the *source* accuracy as the method without adaptation.

- *Continual learning of new tasks:* Five methods of two categories are tested: (1) *regularization-based methods:* Elastic Weight Consolidation (EWC) [54] avoids large changes in model parameters using Fisher information matrix; Adaptive Group Sparsity (AGS) [47] adjusts the regularization strength according to the given task; and Memory Aware Synapses (MAS) [3] determines the regularization strength according to the importance values of model parameters. (2) *Memory-based methods:* Gradient Episodic memory (GEM) [13] calibrates current gradients using gradients calculated on previous stored samples; and Balanced Continual Learning (BCN) [73] controls the strength of learning new task by dynamic programming.
- *Domain adaptation of domain shifts:* Eight methods of three categories are tested: (1) *LayerNorm tuning methods:* SAR [70] only retrains the model parameters in LayerNorm layers. (2) *Feature alignment methods:* Source HypOthesis Transfer (SHOT) [59] retrains the model using entropy minimization and pseudo labels; ConDA [87] improves SHOT by storing previous samples; Bottom-Up Feature Restoration (BUFR) [27] aligns a lightweight feature approximation between domains; Incremental Adversarial Domain Adaptation (IADA) [97] retrains the model with the help of a generative adversarial network; MobileDA [100] retrains the model by distilling knowledge from a larger teach model; and CUA [8] aligns the feature between domains using current and stored samples. (3) *Whole network retraining methods:* Adapting to Changing Environments (ACE) [96] generates labeled images in new domain and use them for retraining.

Offline application-specific adaptation: For each workload, baseline methods use the application's source data to fine-tune the pre-trained FM, and compress it by removing 90% of model parameters. For the six workloads in Table I, all methods use the same hyperparameters: (1) the number of iterations are 80k, 80k, 80k, 10k, 10k, and 80k, respectively; (2) the learning rates are 1e-4, 5e-4, 1e-4, 1e-3, 5e-4, and 1e-4, respectively; (3) the batch sizes are 64, 16, 16, 8, 8, and 64, respectively. The offline training of all methods is executed once on a GPU server (48-GB Quadro RTX 8000 Graphics Card). Baseline methods' training time varies between 3.97 and 24.22 hours, and EdgeTA's training time varies between 8.76 and 46.58 hours.

Online adaptation settings: We use an open-source benchmark [104] that automatically constructs evolving distribution at edge and searches hyperparameters for evaluated methods. The image classification, text classification, and POS tagging workloads are evaluated on Jetson TX2, the semantic segmentation and object detection workloads are evaluated on Jetson Xavier NX, and the visual question answering workload is evaluated on Jetson AGX Xavier. Each workload consists of 30 new input distributions (new tasks or domain shifts). The retraining window is 1800s for image classification and visual question answering workloads with new tasks, and this window is 300s for all other workloads.

In comparison, the proxy model in EdgeTA has the same model size as baseline methods. We report three metrics: (1) *consistent accuracy* [51] over the retraining window; (2) *memory footprint* measured by `jtop` on NVIDIA devices; and (3) *energy consumption* measured by `in_power0_input` interface on devices.

Comparison of accuracy: Fig. 8 illustrates the comparison of consistent accuracy between EdgeTA and baseline methods across different workloads and platforms. We have two key observations from the result. First, EdgeTA consistently achieves higher accuracies than baseline methods at every retraining window. On average, EdgeTA increases accuracy by 21.88% (up to 35.80%). The result indicates that our distribution-adaptive proxy model and continuously updating knowledge base are able to deliver state-of-the-art retraining accuracy under a given resource budget. Second, when running on edge devices with tighter resources (e.g. Jetson TX2 in Figures 8(a)(d)(e)), EdgeTA achieves more improvement in adaptation accuracy. This is because each proxy model consists of the most important neurons to the current input data and has the highest convergency speed in

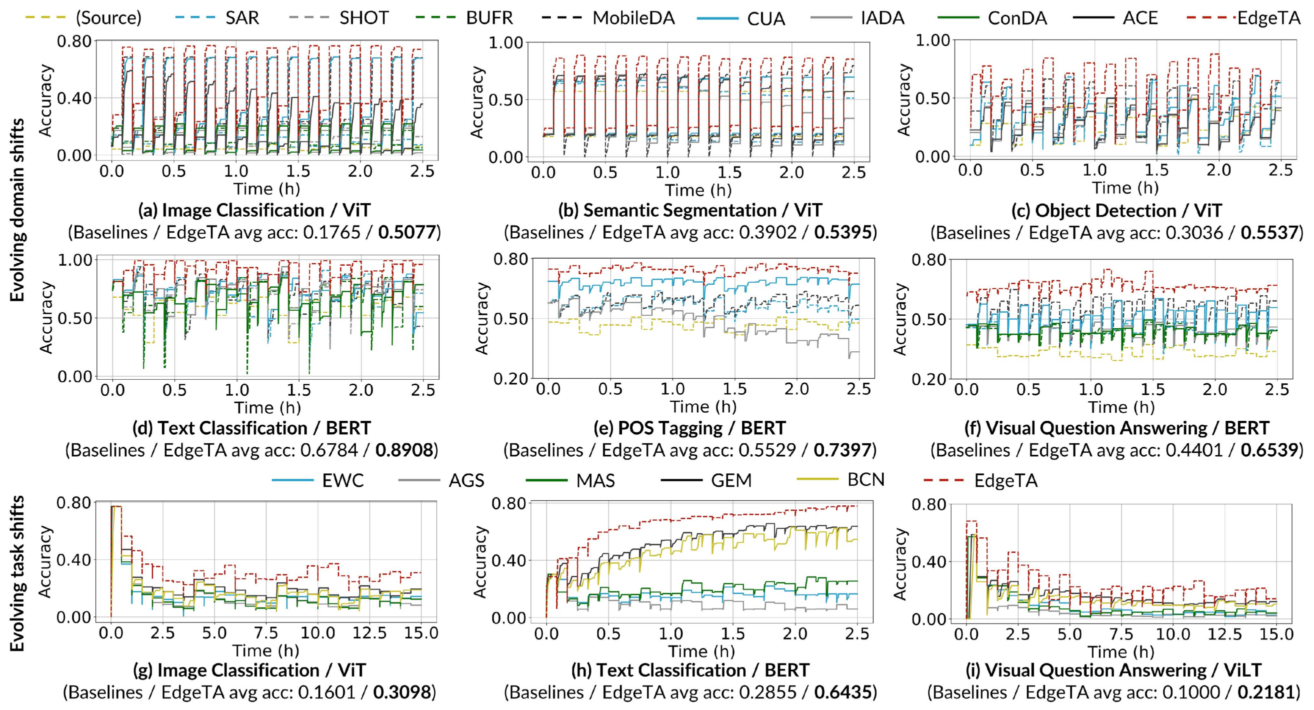


Fig. 8. Accuracy comparison under the evolving distribution.

retraining, thus archiving the largest accuracy improvement per resource unit. In contrast, baseline methods suffer from severe accuracy degradation for two reasons. First, they rely on fixed and compresses models or introduce extra generative networks that are hard to train stably (e.g. IADA and ACE). Second, some methods need to extra regularization computation (e.g. EWC, AGS, and MAS) or process previous samples (GEM and BCN), thus have lower training efficiency within the short retraining window. Overall, EdgeTA improves the accuracy by an average of 21.88% compared to all baselines.

Comparison of memory footprint: Fig. 9 displays the comparison results of all methods' memory footprints over three phases (*workload initialization*, *retraining*, and *retraining completion*) in a retraining window. EdgeTA has two extra phases (*proxy model generation* and *collaborative learning*). We can see that our approach has the lowest memory footprints in most of the phase, because the retraining of its proxy model only processes model neurons/parameters with the largest gradients and the updating of its knowledge base and FM only processes neurons in the linear layers. The only exception is the proxy model generation phase, in which one inference operation on the large knowledge base is performed. This inference completes quickly and this phase is short. In contrast, MobileDA introduces a large teacher model in knowledge distillation and thus has the highest memory usage in most of the cases (Figures 9(a)(b)(d)(f)). BCN concurrently retrains the main model and a reference model and it causes much larger retraining time and memory overhead than other methods. When considering the entire retraining process, EdgeTA achieves 27.14% reductions in memory footprint on average.

TABLE II
AVERAGE ENERGY CONSUMPTION (J) IN EACH DOMAIN SHIFT

	Image Classification	Semantic Segmentation	Object Detection	Text Classification	POS Tagging	Visual Question Answering
SAR	1098.2	874.8	842.6	878.0	726.2	833.0
SHOT	1036.0	-	-	616.7	-	905.3
BUFR	1218.8	-	-	932.4	-	2082.5
MobileDA	1776.0	1478.7	1112.6	1608.4	1678.3	1554.1
CUA	1195.4	916.8	931.7	924.6	893.5	1196.8
IADA	1142.6	935.5	795.8	777.0	808.1	921.5
ConDA	1122.0	-	-	700.0	-	801.0
ACE	2237.8	1741.9	1581.5	-	-	-
EdgeTA	706.4	550.1	458.4	470.9	559.4	718.1

TABLE III
AVERAGE ENERGY CONSUMPTION (J) IN EACH NEW TASK

	Image Classification	Text Classification	Visual Question Answering
EWC	247.4	567.2	2835.7
AGS	386.8	370.0	1749.1
MAS	226.2	555.0	2811.8
GEM	2002.6	1340.3	9329.6
BCN	8414.5	1742.8	14194.3
EdgeTA	165.0	217.9	1287.7

Comparison of energy consumption: Tables II and III list the energy consumption of all methods. The notation "-" in Table II represents the corresponding method is inapplicable for the workload. We can see that EdgeTA consumes the smallest amount of energy because it has the shortest retraining time and the lowest memory usages. In contrast, some baseline methods (e.g. ACE) need long retraining time and some methods (e.g. MobileDA and BCN) need high memory footprints. The later

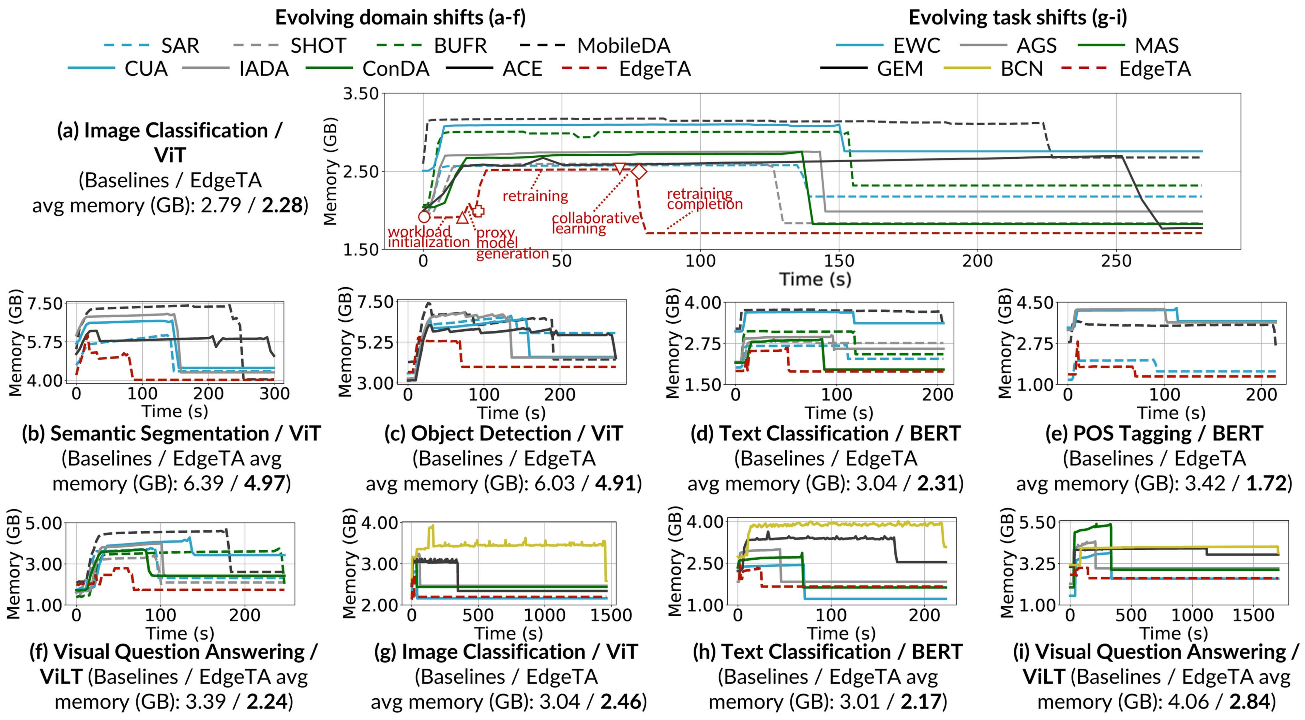


Fig. 9. Memory footprint comparison within one retraining window.

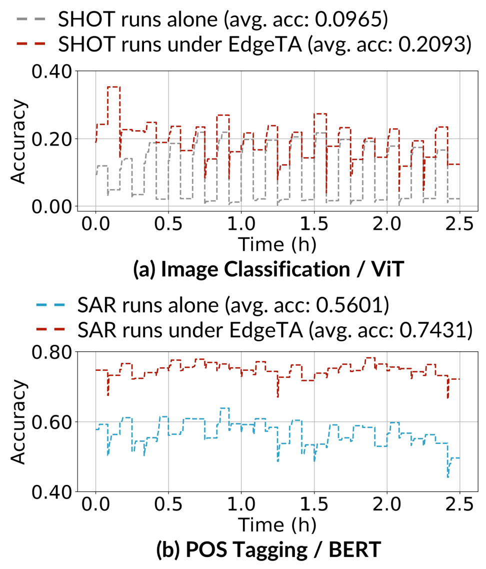


Fig. 10. Accuracies when SHOT and SAR run alone or run under EdgeTA.

methods trigger more page in/out operations between RAM and ROM and thus has higher energy consumption. Compared to all baseline methods, EdgeTA saves energy consumptions by an average of 48.62% and 82.68% in domain adaptation and continual learning jobs, respectively.

Integration with adaptation methods: EdgeTA can use existing adaptation methods to improve their accuracies. We take SHOT (the image classification workload) and SAR (the POS tagging workload) as example and the results in Fig. 10 show that EdgeTA brings 11.28% and 18.30% higher accuracies than the original methods. This is because our proxy model mechanism allows these methods train larger FMs with the same resource budget.

C. Evaluation of Multi-Application Resource Scheduling

Following the setting of the above evaluation, this section evaluates edge resource scheduling in multi-application scenarios. In each scenario, we randomly select 2 to 4 concurrently running applications on an edge device. All tested schedulers aims to maximize these applications' average accuracy in each retraining window (600s).

Compared baselines: We compare EdgeTA with five edge schedulers: (1) *one-time adaptation* [51] only retrains the model in the first distribution; (2) *uniform* [6] retrains the model in each distribution; (3) *AMS* [80] retrains the teacher model and distills its learned knowledge to the compressed model; (4) *Ekya* [6] retrains the model using the best hyperparameters searched by online profiling, and allocates more resources to the application that brings more accuracy improvement; and (5) *RECL* [51] retrains the selected model from a zoo of previous models, and allocates resources like Ekya. The first three schedulers equally allocate resources to each application.

1) Comparison With Edge Resource Schedulers: Fig. 11 demonstrates the fluctuation of model accuracies across different schedulers. Each point in the figure represents the accuracy averaged over all applications. Only domain adaption jobs are tested because AMS, Ekya, and RECL do not support continual learning jobs. The scenarios of Fig. 11(a-b), (c-d), and (e-f) are evaluated on Jetson Xavier NX, Jetson AGX Xavier, and

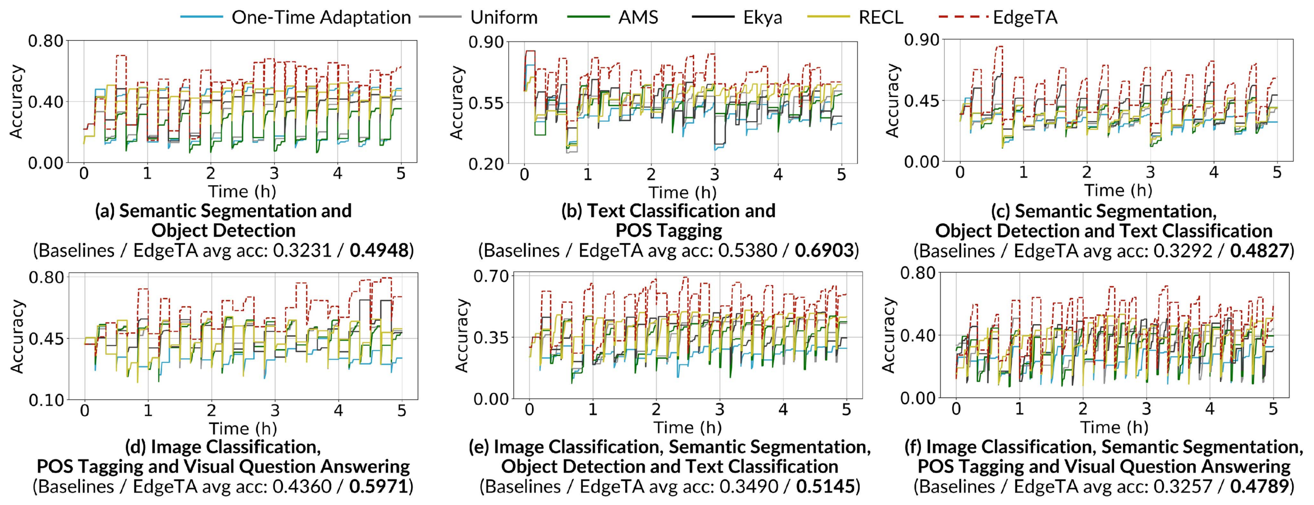


Fig. 11. Comparison of overall accuracy in multi-application scenarios.

Jetson AGX Orin, respectively. We can see EdgeTA achieves consistently higher accuracies than baseline schedulers, because it provides quick estimation of different model sizes' impact on accuracy and first allocates resources to neurons bringing the largest accuracy improvements. In contrast, Ekya needs to perform time-consuming online profiling and less resources can be used in model retraining, thus may result in lower accuracies than the uniform scheduler. RECL adaptively selects an optimal model from the model zoo but stills assumes that all models have the same sizes. Overall, our approach achieves 15.96% improvement in accuracy.

2) Integration With Server-Based Resource Schedulers: In this evaluation, we integrate EdgeTA with three server-based schedulers (AFS [45], Synergy [67], and Muri [107]) that optimize the retraining job's makespan, and discuss how EdgeTA improves their performance. The evaluation uses Fig. 11(e)'s applications, and the three cloud schedulers either run alone (with fixed model sizes) or run with EdgeTA (with neuron-grained resource allocations). The results show that for AFS, Synergy, and Muri, the accuracies of running alone (with fixed model sizes) are 0.4472, 0.3087, and 0.3762; and these accuracies of running with EdgeTA are 0.5198, 0.5248, and 0.5145. Overall, EdgeTA improves server-based schedulers' accuracy by an average of 14.23%.

D. EdgeTA Overhead Breakdown

In a new distribution, EdgeTA's overheads mainly lie in five steps: (1) distribution discrepancy calculation in proxy model estimator; (2) proxy model generation; (3) gradient calculation for identifying retrained neurons; (4) proxy model retraining; (5) FM and knowledge base updating. We evaluate their wall-clock time on the workloads and edge devices of Fig. 8. The results in Fig. 12 show that the proxy model retraining takes most (87.43%) of the wall-clock time, and the total time of other four steps takes only 4% of the retraining window.

Overhead saving of gradient calculation: In identifying the retrained neurons, we compare the computational overhead of

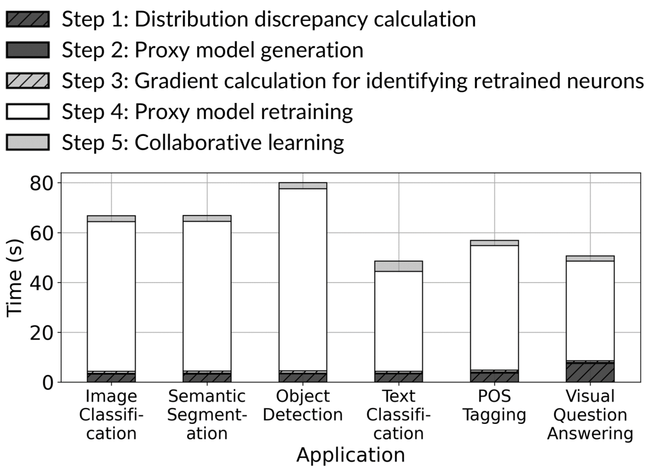


Fig. 12. EdgeTA overhead breakdown at the online stage.

two design choices: calculating the parameters' gradients in all layers (the naive choice) or only the normalization layers (EdgeTA's choice). The results in Fig. 13 show that EdgeTA's choice saves the wall-clock time by 25.23% and the memory footprint by 83.79%. This is because: (1) EdgeTA immediately releases the parameters' gradients of other layers (except normalization layers) after calculation and reduces the memory footprint; (2) the less memory footprint leads to 69.23% less memory allocation operations (measured by PyTorch Profiler), thereby reducing wall-clock time.

E. Understanding EdgeTA's Accuracy Improvements

Using the image classification and semantic segmentation workloads, this section discusses the design choices of EdgeTA. The first evaluation consists of four ablation studies.

- *Knowledge base size:* This size of knowledge base determines the upper bound of the proxy model's accuracy. Fig. 14(a) shows that using the original size (1.00x size) indeed brings the highest accuracy.

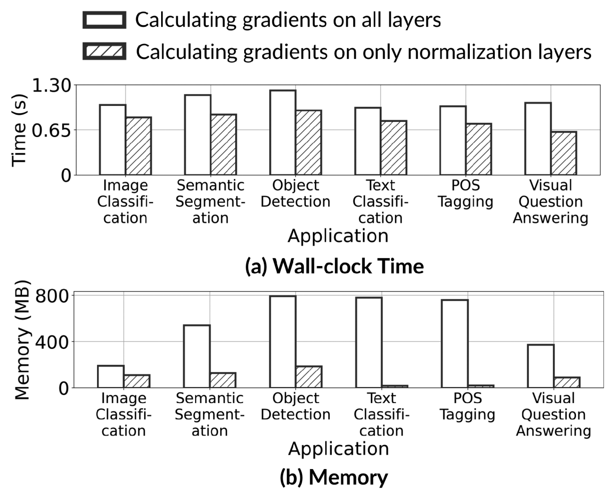


Fig. 13. Overheads of calculating gradients on all layers or only normalization layers.

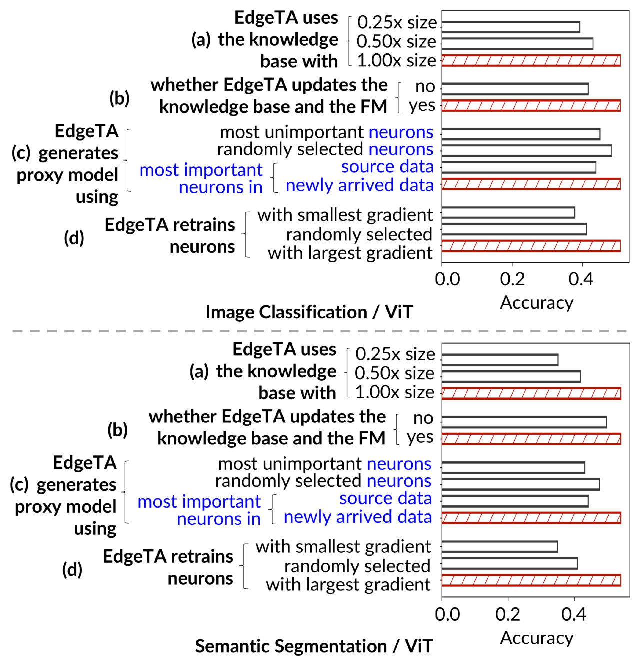


Fig. 14. Accuracies when EdgeTA under different design choices.

- *Updating of FM and knowledge base:* This updating accumulates knowledge from newly arrived input data and Fig. 14(b) shows that it brings 9.12% and 4.33% accuracy improvements in two workloads because of the accumulated knowledge from the previous input data.
- *Proxy model generation strategy:* Fig. 14(c) shows that retaining the neurons of the largest importance values achieves 4.13% and 8.63% accuracy improvements in two workloads, indicating that these neurons indeed contribute to most of model accuracy.
- *Model retraining strategy:* Fig. 14(d) shows that retraining the proxy model with the model gradients of the largest gradients brings 11.32% and 16.07% accuracy improvements.

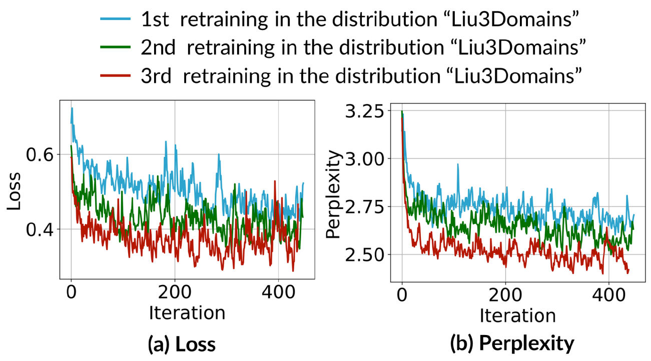


Fig. 15. Loss and model perplexity under the same distribution.

The second evaluation discuss how our proxy model achieves high accuracy using only a small proportion of neurons based on two theories: (1) the *gradient flow theory* [64] states that a model's loss in training is inversely correlated to its model parameters' importance values. EdgeTA's proxy model thus preserves the most important neurons to the newly arrived input data. Fig. 14(c)'s result shows this choice indeed brings 6.72% accuracy improvement. (2) The *winning hand theory* [21] states that the upper bound of a model's accuracy is inversely proportional to the discrepancy between the current input data and the data used to calculate neuron importances. Fig. 14(c) shows using the newly arrived data (i.e. the data used in retraining) rather than the source data (i.e. data used in pre-training) in calculating neurons' importances achieves 8.39% higher accuracy.

The third evaluation uses the POS tagging workload as an example and discusses how the collaborative learning further improves accuracy. Fig. 15 demonstrates the curves of losses and model perplexities when EdgeTA encounters the same input distribution "Liu3Domains" for the first, second, and third time. Model perplexity represents the number of choices when predicting the next word's tag (lower is better). We can see that EdgeTA achieves considerable lower losses and model perplexities when learning the same distribution for the second and third time, because it accumulates the knowledge of seen data in the FM.

F. Discussion of PEFT

In recent years, PEFT techniques have been widely used to efficiently adapt large FMs to a given dataset. Although these techniques only need to tune a small proportion of model parameters in training, they still perform the forward operation on all parameters and take large memory footprint. Hence on resource constrained edge devices, current edge retraining systems still use compressed FMs in adaptation [6], [51]. In this section, we ran a new set of experiments on GPT-Neo [7] to compare CUA (a technique that tunes all model parameters), LoRA [41] (a typical PEFT technique), and EdgeTA on five edge devices: a CPU device Raspberry Pi (4 GB memory), and four GPU devices: NVIDIA Jetson TX2 (8 GB), Xavier NX (16 GB), AGX Xavier (32 GB) and Orin (32 GB). Our comparative evaluation reports three metrics as follows.

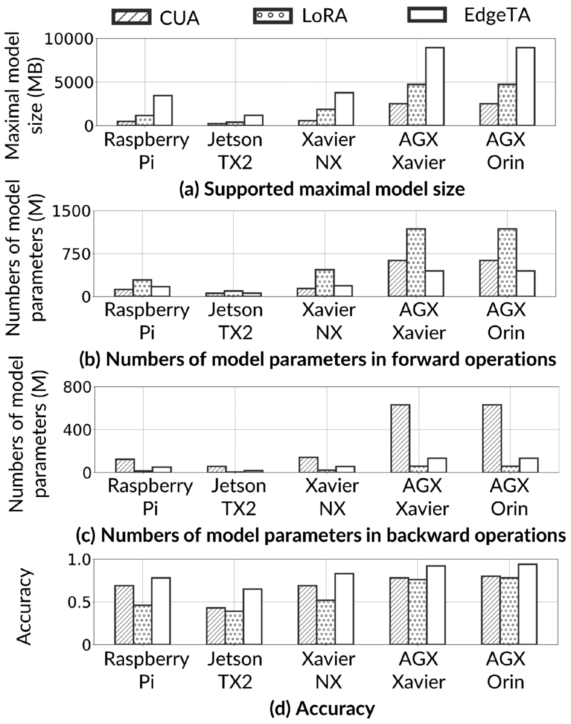


Fig. 16. Comparison of three retraining techniques.

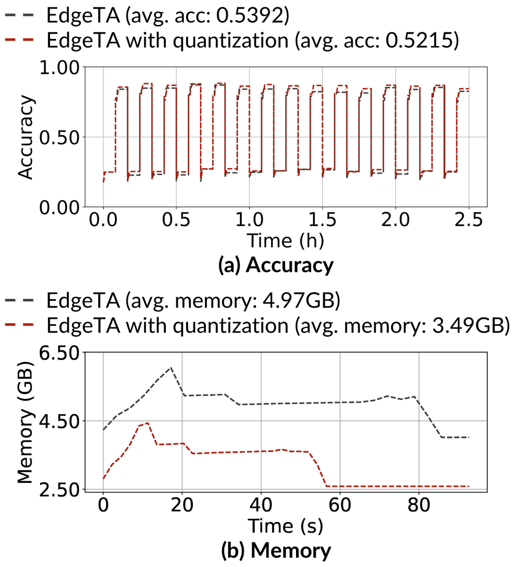


Fig. 17. Accuracy and memory footprint of EdgeTA when it runs alone or runs with quantization.

TABLE IV
ACCURACY AND OVERHEAD OF RETRAINING FM WITH QUANTIZATION AND EDGETA

	Accuracy	Memory	Wall-clock time
Retraining FM (16 bit)	0.5835	17426MB (OOM)	187s
Retraining FM (8 bit)	0.5829	14240MB (OOM)	251s
Retraining FM (4 bit)	0.0411	11094MB	251s
EdgeTA	0.5395	6031MB	78s

TABLE V
SUPPORTED MAXIMAL MODEL SIZE ON FOUR EDGE DEVICES WITH EDGETA

	Jetson TX2	Xavier NX	AGX Xavier	AGX Orin
ViT	1186MB	3772MB	9145MB	9145MB
BERT	1541MB	4752MB	10943MB	10943MB
ViLT	1880MB	5227MB	12392MB	12392MB

The maximal model sizes on edge devices: Fig. 16(a) shows that on the same device, LoRA supports a larger model size (ranging from 392 MB to 4730 MB) than CUA (ranging from 230 MB to 2521 MB). In contrast, EdgeTA can support the largest model sizes (ranging from 1186 MB to 8945 MB). This is because its proxy mechanism first trains a small proxy model and then feedbacks the learned knowledge to the original model layer by layer. Overall, EdgeTA increases model sizes by an average of 2.7x.

Numbers of model parameters in forward/backward operations: Figures 16(b) shows that in forward operations, LoRA processes the largest numbers of model parameters because its model size is larger than that of CUA. In backward operations, Fig. 16(c) shows that CUA updates the largest numbers of model parameters because only this method needs to train the whole model. In contrast, EdgeTA only retrains a small proportion of most important model parameters, thus reducing the numbers of parameters by 1.83x and 2.21x in forward and backward operations, respectively.

Model accuracy with retraining window: Fig. 16(d) shows that within the same retraining window, CUA achieves 10.60% higher accuracy than LoRA because it updates 3.7x more model parameters. EdgeTA achieves 19.4% accuracy improvement compared to LoRA and CUA for two reasons: EdgeTA updates the model parameters that are most important to model accuracy, and it accumulates the learned knowledge in the knowledge base.

G. Discussion of Quantization

Quantization reduces the memory footprint and training time of FMs in pre-training [19]. By integrating with quantization, EdgeTA can also quantize its data structures (knowledge base, proxy model, and neuron indexes) to improve training efficiency. Using the semantic segmentation workload as an example, Fig. 17's evaluation result shows that the 16-bit quantization reduces the memory footprint by 29.78% and the retraining time by 31.32%, while sacrificing only 1.77% accuracy.

We further compare EdgeTA with *the original/uncompressed FM* with quantization. The result in Table IV shows that although retraining the original FM with 16-bit or 8-bit quantization has higher accuracies than EdgeTA, it needs 20x larger memory and training wall-clock time than our approach. Retraining the original FM with 4-bit quantization fails to converge. In conclusion, directly training large FMs with quantization either incurs Out-Of-Memory (OOM) errors on resource-constrained edge devices or significantly degrades model accuracy.

H. Upper Bound of Model Size in EdgeTA

Table V lists the maximal sizes of three typical transformers when running on different devices with EdgeTA. We can see the upper bound of a model is determined by the device's available memory and the model's network architecture. This is because different neural networks have different maximal compression ratio, e.g. 1.45% for ViT, 1.14% for BERT, and 1.10% for ViLT.

VI. Related Work

Elastic model scaling for stationary input distributions: On mobile and edge devices, scaling model sizes at run-time to make trade-offs between inference latency and model accuracy has been a hot topic recently [12], [101], [102]. Current dynamic network methods either construct networks (e.g. MSDNet [44]) that allow early exit from an inference task after meeting accuracy targets [24], [56], [62], [88], [89], or generate a list of descendent models of different sizes (e.g. nested networks [28], [40], [93] or network blocks [33]) and dynamically select one of them according to the available resource. However, both constructed networks and descendent models are produced for a *stationary* distribution and hence may suffer from large accuracy depredations when the new distribution arrives. Similarly, based on a stationary input distribution, neural architecture search (NAS) techniques select models from a DNN pool [48], [63] or automatically search network architectures [84], [108] to optimize their accuracies according to the underlying devices.

FM adaptation/retraining at edge: Deploying a FM on an edge device requires both application-specific adaptation (full/resource/data-efficient fine-tuning [11], [14], [38] or linear evaluation [72]) and model compression (parameter pruning [30], [35], [53], [57] or knowledge distillation [90]). Current deployment methods [77], [79], [103] take hours to complete even using powerful GPU servers or multiple GPU edge devices [4], [55], [58], [68], [69]. Hence when encountering evolving input data at edge, existing adaptation techniques, namely continual learning [3], [13], [47], [54], [73], [99] and domain adaptation [8], [27], [59], [70], [87], [96], [97], [100], directly retrain the compressed model and thus cannot excavate the FM's scaling potential.

Edge-based scheduling systems: Early edge-based schedulers focus on the inference jobs and optimize their latency, accuracy or energy consumption [5], [28], [34], [46], [65], [106]. This run-time optimization is based on the offline generation and profiling of multiple descendant/nested models [28] or network blocks [33]. Latest edge schedulers further consider a mix of inference and retraining tasks. Specifically, Ekya [6] dynamically manages hyper-parameters (e.g. batch size and learning rate) in retraining tasks. RECL [51] first selects the best model from a pool of history/seen models when an input distribution shift occurs and then optimizes resource allocations among multiple models. However, existing edge schedulers assume that the model sizes are fixed in retraining tasks, hence limiting the decision space and overlooking the optimal accuracy tradeoff.

VII. Conclusion

This paper presents EdgeTA, a neuron-grained FM scaling system to improve model retraining performance at edge. Our approach can support distribution adaptive generation of small proxy model to achieve both high accuracy and low overheads in learning new data. At run-time, EdgeTA can search the optimal model sizes and allocated resources of multiple applications to maximize their overall accuracy. Extensive evaluation results prove the efficacy and practicality of EdgeTA.

References

[1] 1995. [Online]. Available: http://people.csail.mit.edu/jrennie/20Newsgroups/20news-bydate.tar.gz
[2] Lightweight kubernetes. 2018. [Online]. Available: https://k3s.io
[3] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 144–161.
[4] G. Ayache and S. E. Rouayheb, "Private weighted random walk stochastic gradient descent," *IEEE J. Sel. Areas Inf. Theory*, vol. 2, no. 1, pp. 452–463, Mar. 2021.
[5] S. Bateni and C. Liu, "NeuOS: A latency-predictable multi-dimensional optimization framework for DNN-driven autonomous systems," in *Proc. USENIX Annu. Techn. Conf.*, 2020, pp. 371–385.
[6] R. Bhardwaj et al., "Ekya: Continuous learning of video analytics models on edge compute servers," in *Proc. Symp. Netw. Syst. Des. Implementation*, 2022, pp. 119–135.
[7] S. Black, L. Gao, P. Wang, C. Leahy, and S. Biderman, "GPT-Neo: Large scale autoregressive language modeling with mesh-tensorflow," Zenodo, version 1.0, Mar. 2021, doi: 10.5281/zenodo.5297715.
[8] A. Bobu, E. Tzeng, J. Hoffman, and T. Darrell, "Adapting to continuously shifting domains," in *Proc. Int. Conf. Learn. Representations Workshop*, 2018, pp. 119–135.
[9] R. Bommasani et al., "On the opportunities and risks of foundation models," *arXiv:2108.07258*, 2021.
[10] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "FreezeOut: Accelerate training by progressively freezing layers," 2017, *arXiv: 1706.04983*.
[11] T. Brown et al., "Language models are few-shot learners," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 1877–1901.
[12] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," in *Proc. Int. Conf. Learn. Representations*, 2020, pp. 1–14.
[13] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, "Efficient lifelong learning with A-GEM," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–12.
[14] S. Chen et al., "AdaptFormer: Adapting vision transformers for scalable visual recognition," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2022, pp. 16664–16678.
[15] H. Cheng, M. Zhang, and J. Q. Shi, "A survey on deep neural network pruning-taxonomy, comparison, analysis, and recommendations," 2023, *arXiv:2308.06767*.
[16] M. Cordts et al., "The cityscapes dataset for semantic urban scene understanding," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 3213–3223.
[17] D. Dai, L. Dong, Y. Hao, Z. Sui, B. Chang, and F. Wei, "Knowledge neurons in pretrained transformers," in *Proc. Conf. Assoc. Comput. Linguistics*, 2022, pp. 8493–8502.
[18] M. D. Lange et al., "A continual learning survey: Defying forgetting in classification tasks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 7, pp. 3366–3385, Jul. 2022.
[19] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "GPT3.int8(): 8-bit matrix multiplication for transformers at scale," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2022, pp. 30318–30332.
[20] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Assoc. Comput. Linguistics: Hum. Lang. Technol.*, 2019, pp. 4171–4186.
[21] J. Diffenderfer, B. R. Bartoldson, S. Chaganti, J. Zhang, and B. Kailkhura, "A winning hand: Compressing deep networks can improve out-of-distribution robustness," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2021, pp. 664–676.
[22] X. Ding, G. Ding, Y. Guo, and J. Han, "Centripetal SGD for pruning very deep convolutional networks with complicated structure," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4943–4953.
[23] X. Ding, B. Liu, and P. S. Yu, "A holistic lexicon-based approach to opinion mining," in *Proc. ACM Int. Conf. Web Search Data Mining*, 2008, pp. 231–240.

[24] F. Dong et al., "Multi-exit DNN inference acceleration based on multi-dimensional optimization for edge intelligence," *IEEE Trans. Mobile Comput.*, vol. 22, no. 9, pp. 5389–5405, Sep. 2023.

[25] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Representations*, 2021, pp. 1–12.

[26] D. C. Dowson and B.V. Landau, "The fréchet distance between multivariate normal distributions," *J. Multivariate Anal.*, vol. 12, no. 3, pp. 450–455, 1982.

[27] C. Eastwood, I. Mason, C. K. I. Williams, and B. Schölkopf, "Source-free adaptation to measurement shift via bottom-up feature restoration," in *Proc. Int. Conf. Learn. Representations*, 2022, pp. 1–14.

[28] B. Fang, X. Zeng, and M. Zhang, "NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proc. Annu. Int. Conf. Mobile Comput. Netw.*, 2018, pp. 115–127.

[29] D. Ganguli et al., "Predictability and surprise in large generative models," in *Proc. ACM Conf. Fairness, Accountability, Transparency*, 2022, pp. 1747–1764.

[30] X. Gao, Y. Zhao, L. Dudziak, R. D. Mullins, and C.-Z. Xu, "Dynamic channel pruning: Feature boosting and suppression," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–11.

[31] Y. Goyal, T. Khot, D. Summers-Stay, D. Batra, and D. Parikh, "Making the v in VQA matter: Elevating the role of image understanding in visual question answering," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 6325–6334.

[32] X. Guo and H. Yu, "On the domain adaptation and generalization of pretrained language models: A survey," 2022, *arXiv:2211.03154*.

[33] R. Han, Q. Zhang, C. H. Liu, G. Wang, J. Tang, and L. Y. Chen, "LegoDNN: Block-grained scaling of deep neural networks for mobile vision," in *Proc. Annu. Int. Conf. Mobile Comput. Netw.*, 2021, pp. 406–419.

[34] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "MCDNN: An approximation-based execution framework for deep stream processing under resource constraints," in *Proc. 14th Annu. Int. Conf. Mobile Syst., Appl., Serv.*, 2016, pp. 123–136.

[35] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learn. Representations*, 2015, pp. 1–14.

[36] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic neural networks: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 11, pp. 7436–7456, Nov. 2022.

[37] J. He, C. Zhou, X. Ma, T. Berg-Kirkpatrick, and G. Neubig, "Towards a unified view of parameter-efficient transfer learning," in *Proc. Int. Conf. Learn. Representations*, 2022, pp. 1–12.

[38] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. B. Girshick, "Masked autoencoders are scalable vision learners," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 15979–15988.

[39] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–11.

[40] L. Hou, Z. Huang, L. Shang, X. Jiang, X. Chen, and Q. Liu, "DynaBERT: Dynamic BERT with adaptive width and depth," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 9782–9793.

[41] E. J. Hu et al., "LoRA: Low-rank adaptation of large language models," in *Proc. Int. Conf. Learn. Representations*, 2022, pp. 1–16.

[42] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-excitation networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 8, pp. 2011–2023, Aug. 2020.

[43] M. Hu and B. Liu, "Mining and summarizing customer reviews," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2004, pp. 168–177.

[44] G. Huang, D. Chen, T. Li, F. Wu, L. V. D. Maaten, and K. Q. Weinberger, "Multi-scale dense convolutional networks for efficient prediction," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–12.

[45] C. Hwang, T. Kim, S. Kim, J. Shin, and K. Park, "Elastic resource sharing for distributed deep learning," in *Proc. Symp. Netw. Syst. Des. Implementation*, 2021, pp. 721–739.

[46] A. H. Jiang et al., "Mainstream: Dynamic stem-sharing for multi-tenant video processing," in *Proc. USENIX Annu. Techn. Conf.*, 2018, pp. 29–42.

[47] S. Jung, H. Ahn, S. Cha, and T. Moon, "Continual learning with node-importance based adaptive group sparse regularization," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 3647–3658.

[48] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "No-Scope: Optimizing neural network queries over video at scale," in *Proc. VLDB Endowment*, vol. 10, no. 11, 2017, pp. 1586–1597.

[49] J. Kaplan et al., "Scaling laws for neural language models," 2020, *arXiv: 2001.08361*.

[50] M. G. Kendall, "A. new measure of rank correlation," *Biometrika*, vol. 30, no. 1/2, pp. 81–93, 1938.

[51] M. Khani et al., "RECL: Responsive resource-efficient continuous learning for video analytics," in *Proc. Symp. Netw. Syst. Des. Implementation*, 2023, pp. 917–932.

[52] W. Kim, B. Son, and I. Kim, "ViLT: Vision-and-language transformer without convolution or region supervision," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 5583–5594.

[53] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," in *Proc. Int. Conf. Learn. Representations*, 2016, pp. 1–11.

[54] J. Kirkpatrick et al., "Overcoming catastrophic forgetting in neural networks," in *Proc. Nat. Acad. Sci.*, vol. 114, no. 13, pp. 3521–3526, 2017.

[55] A. Koloskova, S. U. Stich, and M. Jaggi, "Decentralized stochastic optimization and gossip algorithms with compressed communication," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 3478–3487.

[56] E. Li, Z. Zhou, and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," in *Proc. Workshop Mobile Edge Commun.*, 2018, pp. 31–36.

[57] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–12.

[58] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 3049–3058.

[59] J. Liang, D. Hu, and J. Feng, "Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 6028–6039.

[60] T.-Y. Lin et al., "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 740–755.

[61] Q. Liu, Z. Gao, B. Liu, and Y. Zhang, "Automated rule selection for aspect extraction in opinion mining," in *Proc. Int. Joint Conf. Artif. Intell.*, 2015, pp. 1291–1297.

[62] Z. Liu et al., "Hastening stream offloading of inference via multi-exit DNNs in mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 23, no. 1, pp. 535–548, Jan. 2024.

[63] B. Lu, J. Yang, L. Y. Chen, and S. Ren, "Automating deep neural network model selection for edge inference," in *Proc. IEEE 1st Int. Conf. Cogn. Mach. Intell.*, 2019, pp. 184–193.

[64] E. SinghLubana and R. P. Dick, "A gradient flow framework for analyzing network pruning," in *Proc. Int. Conf. Learn. Representations*, 2021, pp. 1–10.

[65] A. Mathur, N. D. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, and F. Kawsar, "DeepEye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware," in *Proc. 15th Annu. Int. Conf. Mobile Syst., Appl., Serv.*, 2017, pp. 68–81.

[66] S. Mehta and M. Rastegari, "MobileViT: Light-weight, general-purpose, and mobile-friendly vision transformer," in *Proc. Int. Conf. Learn. Representations*, 2022, pp. 1–13.

[67] J. Mohan, A. Phanishayee, J. Kulkarni, and V. Chidambaram, "Looking beyond GPUs for DNN scheduling on multi-tenant clusters," in *Proc. USENIX Conf. Operating Syst. Des. Implementation*, 2022, pp. 579–596.

[68] G. Nadiradze, A. Sabour, P. Davies, S. Li, and D. Alistarh, "Asynchronous decentralized SGD with quantized and local updates," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2021, pp. 6829–6842.

[69] A. Nedic and A. E. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Trans. Autom. Control.*, vol. 54, no. 1, pp. 48–61, Jan. 2009.

[70] S. Niu et al., "Towards stable test-time adaptation in dynamic wild world," in *Proc. Int. Conf. Learn. Representations*, 2023, pp. 1–14.

[71] V. Prabhu, A. Chandrasekaran, K. Saenko, and J. Hoffman, "Active domain adaptation via clustering uncertainty-weighted embeddings," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2021, pp. 8505–8514.

[72] A. Radford et al., "Learning transferable visual models from natural language supervision," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 8748–8763.

[73] K. Raghavan and P. Balaprakash, "Formalizing the generalization-forgetting trade-off in continual learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2021, pp. 17284–17297.

[74] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "ZeRo: Memory optimizations toward training trillion parameter models," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2020, pp. 1–16.

[75] S. Ren and K. Q. Zhu, "Leaner and faster: Two-stage model compression for lightweight text-image retrieval," in *Proc. Conf. North Amer. Assoc. Comput. Linguistics*, 2022, pp. 4085–4090.

[76] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, “Playing for data: Ground truth from computer games,” in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 102–118.

[77] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter,” 2019, *arXiv: 1910.01108*.

[78] J. Schneider, “Foundation models in brief: A historical, socio-technical focus,” 2022, *arXiv:2212.08967*.

[79] G. Shin, W. Xie, and S. Albanie, “NamedMask: Distilling segmenters from complementary foundation models,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2023, pp. 4960–4969.

[80] M. K. Shirkoohi, P. Hamadanian, A. Nasr-Esfahany, and M. Alizadeh, “Real-time video inference on edge devices via adaptive model streaming,” in *Proc. IEEE Int. Conf. Comput. Vis.*, 2021, pp. 4552–4562.

[81] Y. Song, Z. Mi, H. Xie, and H. Chen, “PowerInfer: Fast large language model serving with a consumer-grade GPU,” 2023, *arXiv:2312.12456*.

[82] B. Sorscher, R. Geirhos, S. Shekhar, S. Ganguli, and A. Morcos, “Beyond neural scaling laws: Beating power law scaling via data pruning,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2022, pp. 19523–19536.

[83] X. Sun, P. Zhang, P. Zhang, H. Shah, K. Saenko, and X. Xia, “DIME-FM : Distilling multimodal and efficient foundation models,” in *Proc. IEEE Int. Conf. Comput. Vis.*, 2023, pp. 15475–15487.

[84] M. Tan et al., “MnasNet: Platform-aware neural architecture search for mobile,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2815–2823.

[85] D. Tang, B. Qin, and T. Liu, “Aspect level sentiment classification with deep memory network,” in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2016, pp. 214–224.

[86] R. Tang, Y. Lu, L. Liu, L. Mou, O. Vechtomova, and J. Lin, “Distilling task-specific knowledge from BERT into simple neural networks,” 2019, *arXiv: 1903.12136*.

[87] A. M.N. Taufique, C. S. Jahan, and A. E. Savakis,”, “CONDA: Continual unsupervised domain adaptation,” 2021, *arXiv:2212.00621*.

[88] S. Teerapittayanon, B. McDanel, and H. T. Kung, “BranchyNet: Fast inference via early exiting from deep neural networks,” in *Proc. 23rd Int. Conf. Pattern Recognit.*, 2016, pp. 2464–2469.

[89] S. Teerapittayanon, B. McDanel, and H. T. Kung, “Distributed deep neural networks over the cloud, the edge and end devices,” in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 328–339.

[90] Y. Tian, D. Krishnan, and P. Isola, “Contrastive representation distillation,” in *Proc. Int. Conf. Learn. Representations*, 2020, pp. 1–13.

[91] A. Vaswani et al., “Attention is all you need,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[92] J. Wei et al., “Emergent abilities of large language models,” 2022, *arXiv:2206.07682*.

[93] H. Wen et al., “AdaptiveNet: Post-deployment neural architecture adaptation for diverse edge environments,” in *Proc. Annu. Int. Conf. Mobile Comput. Netw.*, 2023, pp. 1–17.

[94] T. Wolf et al., “HuggingFace’s transformers: State-of-the-art natural language processing,” 2019, *arXiv: 1910.03771*.

[95] Z. Wu, Y. Huang, Y. Yu, L. Wang, and T. Tan, “Early hierarchical contexts learned by convolutional networks for image segmentation,” in *Proc. Int. Conf. Pattern Recognit.*, 2014, pp. 1538–1543.

[96] Z. Wu, X. Wang, J. E. Gonzalez, T. Goldstein, and L. S. Davis, “ACE: Adapting to changing environments for semantic segmentation,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2121–2130.

[97] M. Wulfmeier, A. Bewley, and I. Posner, “Incremental adversarial domain adaptation for continually changing environments,” in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 4489–4495.

[98] S. Xu et al., “Towards efficient task-driven model reprogramming with foundation models,” 2023, *arXiv:2304.02263*.

[99] B. Yang et al., “EdgeFM: Leveraging foundation model for open-set learning on the edge,” in *Proc. 21st ACM Conf. Embedded Netw. Sensor Syst.*, 2023, pp. 111–124.

[100] J. Yang, H. Zou, S. Cao, Z. Chen, and L. Xie, “MobileDA: Toward edge-domain adaptation,” *IEEE Internet Things J.*, vol. 7, no. 8, pp. 6909–6918, Aug. 2020.

[101] J. Yu and T. S. Huang, “Universally slimmable networks and improved training techniques,” in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019, pp. 1803–1811.

[102] J. Yu, L. Yang, N. Xu, J. Yang, and T. S. Huang, “Slimmable neural networks,” in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–11.

[103] M. Zhang, U.-N. Niranjan, and Y. He, “Adversarial data augmentation for task-specific knowledge distillation of pre-trained transformers,” in *Proc. AAAI Conf. Artif. Intell.*, 2022, pp. 11685–11693.

[104] Q. Zhang, R. Han, C. H. Liu, G. Wang, and L. Y. Chen, “EdgeVision-Bench: A benchmark of evolving input domains for vision applications at edge,” in *Proc. IEEE 39th Int. Conf. Data Eng.*, 2023, pp. 3643–3646.

[105] Z. Zhang, B. Guo, W. Sun, Y. Liu, and Z. Yu, “Cross-FCL: Toward a cross-edge federated continual learning framework in mobile edge computing systems,” *IEEE Trans. Mobile Comput.*, vol. 23, no. 1, pp. 313–326, Jan. 2024.

[106] K. Zhao et al., “EdgeAdaptor: Online configuration adaption, model selection and resource provisioning for edge DNN inference serving at scale,” *IEEE Trans. Mobile Comput.*, vol. 22, no. 10, pp. 5870–5886, Oct. 2023.

[107] Y. Zhao, Y. Liu, Y. Peng, Y. Zhu, X. Liu, and X. Jin, “Multi-resource interleaving for deep learning training,” in *Proc. Conf. ACM Special Int. Group Data Commun.*, 2022, pp. 428–440.

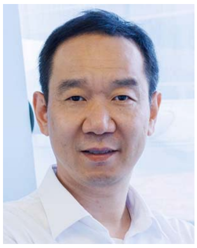
[108] B. Zoph and V. Q. Le, “Neural architecture search with reinforcement learning,” in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–14.

Qinglong Zhang is working toward thne master’s degree with the School of Computer Science and Technology, Beijing Institute of Technology. His work focuses on edge intelligence and deep learning applications.

Rui Han received the MSc degree with honor from Tsinghua University, China, in 2010, and the PhD degree from Imperial College London, U.K., in 2014. He is an associate professor with the School of Computer Science and Technology, Beijing Institute of Technology, China. Before joining BIT, His research interests are system optimization for cloud data center workloads (in particular highly parallel services and deep learning applications). He has more than 40 publications in these areas, including papers at MobiCOM, *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Knowledge and Data Engineering*, INFOCOM, and ICDCS.

Chi Harold Liu (Senior Member, IEEE) received the BEng degree from Tsinghua University, Beijing, China, and the PhD degree from the Imperial College London, London, U.K. He is currently a full professor and the vice dean with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing. Before that, he worked for IBM Research - China and Deutsche Telekom Laboratories, Berlin, Germany, and IBM T. J. Watson Research Center, USA. He is now an associate editor for *IEEE Trans. Network Science and Engineering*. His current research interests include the Big Data analytics, mobile computing, and deep learning. Dr. Liu is a fellow of IET, and a fellow of Royal Society of the Arts.

Guoren Wang (Senior Member, IEEE) received the BSc, MSc, and PhD degrees from the Department of Computer Science, Northeastern University, Shenyang, China, in 1988, 1991, and 1996, respectively. He is now a full professor and director with the Institute of Data Science and Knowledge Engineering, Department of Computer Science and Technology, Beijing Institute of Technology, Beijing, China. He was an assistant president for Northeastern University, China. His research interests include XML data management, query processing and optimization, bioinformatics, high dimensional indexing, parallel database systems, and cloud data management. He has published more than 150 research papers in top conferences and journals like *IEEE Transactions on Knowledge and Data Engineering*, ICDE, SIGMOD, VLDB, etc.



Song Guo (Fellow, IEEE) is a full professor with the Department of Computer Science and Engineering (CSE), Hong Kong University of Science and Technology (HKUST). Before joining HKUST in 2023, he was a professor with The Hong Kong Polytechnic University. He also holds a Changjiang Chair Professorship awarded by the Ministry of Education of China. Hre is a fellow with the Canadian Academy of Engineering. His research interests are mainly in edge AI, machine learning, mobile computing, and distributed systems. He published many papers in top venues with wide impact in these areas and was recognized as a Highly Cited Researcher (Clarivate Web of Science). He is the recipient of over a dozen Best Paper Awards from IEEE/ACM conferences, journals, and technical committees. PHe is the editor-in-chief of *IEEE Open Journal of the Computer Society* and the chair of IEEE Communications Society (ComSoc) Space and Satellite Communications Technical Committee. He was an IEEE ComSoc distinguished lecturer and a member of IEEE ComSoc Board of Governors. He has served for IEEE Computer Society on Fellow Evaluation Committee, and been named on editorial board of a number of prestigious international journals like *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Emerging Topics in Computing*, etc.

Lydia Y. Chen (Senior Member, IEEE) received the BA degree from National Taiwan University, and the PhD degree from the Pennsylvania State University. She is an associate professor with the Department of Computer Science, Technology University Delft. Prior to joining TU Delft, she was a research staff member with the IBM Zurich Research Lab from 2007 to 2018. Her research interests center around dependability management, resource allocation and privacy enhancement for large scale data processing systems and services. She has published more than 80 papers in journals, e.g., *IEEE Transactions on Distributed Systems*, *IEEE Transactions on Service Computing*, and conference proceedings, e.g., INFOCOM, Sigmetrics, DSN, and Eurosys. She was a co-recipient of the best paper awards at CCgrid'15 and eEnergy'15.